UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

FACULTAD DE INGENIERÍA ELÉCTRICA

"REST API PARA RED SOCIAL DEL NEGOCIO AGRÍCOLA"

TESIS

QUE PARA OBTENER EL TÍTULO DE
INGENIERO EN COMPUTACIÓN

PRESENTA
JESÚS ALEJANDRO SERRATO PANIAGUA

ASESOR
MAESTRO EN INGENIERÍA OPCIÓN SISTEMAS COMPUTACIONALES SAMUEL PÉREZ AGUILAR

ASESORES EXTERNOS
DOCTORA EN CIENCIAS EN NEGOCIOS INTERNACIONALES ODETTE VIRGINIA DELFÍN ORTEGA

INGENIERO ELECTRICISTA RAMIRO SERRATO PANIAGUA

MORELIA, MICHOACÁN
JULIO 2014

Dedicatoria

A mi padre
Ramiro Serrato
quien ha dedicado su vida entera a su familia y
ha sido el sustento y el pilar de la misma.

A mi madre
María H. Paniagua
quien siempre ha estado allí para sus hijos
brindando su apoyo y amor incondicional en todo momento.

A mi hermano
Ramiro Serrato
que siempre me ha enseñado cosas nuevas y que me ha
apoyado y guiado en la realización de esta Tesis en todo momento.

A mis hermanas
Belinda Y. Serrato
Diana P. Serrato
quienes me han enseñado que el mundo está lleno
de arte y que en esta vida siempre
hay algo nuevo por aprender.

Agradecimientos

A mi familia, porque siempre me apoyaron para que pudiera terminar mi carrera profesional y esta Tesis.

A mi asesor de Tesis, el M.I. Samuel Pérez Aguilar, quien me apoyo en la realización de esta Tesis.

A mis asesores externos de Tesis:
La Dra. Odette V. Delfín Ortega, que me asesoró en un área completamente desconocida para mí y brindó un enfoque distinto a esta Tesis.
El Ing. Ramiro Serrato Paniagua, quien me brindo asesoría y conocimiento a lo largo de este proyecto de Tesis

A mis amigos y compañeros, quienes me acompañaron a lo largo de este largo viaje y con los cuales vivimos momentos inolvidables y que me inspiraron a continuar y a terminar este trabajo.

A todos los maestros, que me enseñaron cosas nuevas y valiosas y a los que no, porque también fueron parte de mi desarrollo.

A la Facultad de Ingeniería Eléctrica, porque fue durante mi estancia en ella que obtuve mi formación en ingeniería.

A todos ellos:
¡Muchas gracias!

Contenido

Lista de Figuras

Lista de Tablas

Abreviaturas

REST	Representational State Transfer
API	Application Programming Interface (Interfaz de Programación de Aplicaciones)
Java EE	Java Enterprise Edition (Edición Empresarial de Java)
UML	Unified Modeling Language (Lenguaje Unificado de Modelado)
MVC	Model View Controller (Modelo Vista Controlador)
DAO	Data Access Object (Objeto de Acceso a Datos)
RFQ	Request For Quotation (Solicitud de cotización)
IoC	Inversion of Control (Inversión de Control)
E-R	Entidad-Relación
cXML	commerce Extensible Markup Language (Lenguaje de Marcado Extensible de Comercio)
USDA	United States Department of Agriculture (Departamento de Agricultura de los Estados Unidos)
OECD	Organisation for Economic Co-operation and Development (Organización para la Cooperación y el Desarrollo Económicos)
FAO	Food and Agriculture Organization of the United Nations (Organización de las Naciones Unidas para la alimentación y la Agricultura)
U.S.	United States (Estados Unidos)
HTML	HyperText Markup Language (Lenguaje de Marcas de Hipertexto)
PCA	Principal Component Analysis (Análisis de Componentes Principales)
HSQLDB	HyperSQL DataBase
NoSQL	Not only SQL
SQL	Structured Query Language (Lenguaje de Consulta Estructurado)
BSD	Berkeley Software Distribution (Distribución de Software Berkeley)
ANSI	American National Standards Institute (Instituto Nacional Estadounidense de Estándares)
JSON	JavaScript Object Notation (Notación de Objeto JavaScript)
ORM	Object Relational Mapping (Mapeo Objeto Relacional)
POO	Programación Orientada a Objetos
JPA	Java Persistence API (API de Persistencia Java)
POJO	Plain Old Java Object (Objeto Java Plano)
JPQL	Java Persistence Query Language
XML	Extensible Markup Language (Lenguaje Extensible de Marcado)
Java SE	Java Standard Edition (Edición Estándar de Java)
EJB	Enterprise JavaBeans (Objetos Java Empresariales)
JSR	Java Specification Requests (Solicitudes de Especificación de Java)
LGPL	GNU Lesser General Public License (Licencia Pública General Reducida de GNU)
HQL	Hibernate Query Language (Lenguaje de Consultas de Hibernate)
HTTP	HyperText Transfer Protocol (Protocolo de Transferencia de Hipertexto)
URI	Uniform Resource Identifier (Identificador de Recursos Uniforme)
SOAP	Simple Object Access Protocol (Protocolo Simple de Acceso a Objetos)
JVM	Java Virtual Machine (Máquina Virtual de Java)
JDBC	Java DataBase Connectivity (Conectividad a Base de Datos de Java)

JTA	Java Transaction API (API para Transacciones de Java)
JMS	Java Message Service (Servicio de Mensajes Java)
DI	Dependency Injection (Inyección de Dependencias)
JSP	JavaServer Pages (Páginas Java de Servidor)
JSTL	JavaServer Pages Standard Tag Library (Librería Estándar de Etiquetas de Página Java de Servidor)
SHA	Secure Hash Algorithm (Algoritmo de Hash Seguro)
MD5	Message Digest 5 Algorithm (Algoritmo de Resumen del Mensaje 5)
MongoDB	Mongo (de la palabra en inglés “Humongous”) DataBase
SoC	Separation of Concerns (Separación de responsabilidad)
IDE	Integrated Development Environment (Ambiente de Desarrollo Integrado)
MIT	Massachusetts Institute of Technology (Instituto de Tecnología de Massachusetts)
URL	Uniform Resource Locator (Localizador de Recursos Uniforme)
DTO	Data Transfer Object (Objeto de Transferencia de Datos)
BD	Base de Datos
DB	DataBase (Base de Datos)
IE	Information Engineering (Ingeniería de la Información)
BSON	Binary JSON

Resumen

El difícil acceso a los mercados agrícolas externos y locales y la carencia de herramientas informáticas que faciliten la comunicación entre los diferentes participantes del negocio agrícola se ha tornado en una problemática que aqueja a este sector.

Por otro lado, las redes sociales se han convertido en medios de acceso a la información que nos mantienen informados de cualquier suceso en la actualidad y nos permiten conocer sobre las actividades sociales de las personas y son medios que en la actualidad siguen creciendo y cada día surgen nuevas plataformas y servicios que brindan diferentes funcionalidades a sus usuarios.

El presente trabajo de Tesis propone la introducción de una plataforma social enfocada específicamente al negocio agrícola por medio del desarrollo de un REST API. El REST API el cual se define como una aplicación web que está conformada por un conjunto de servicios web que deben ser consumidos por una aplicación ya sea web, móvil, de escritorio o de cualquier otro tipo. La ventaja del REST API es precisamente que no está atado a una tecnología en específico se trata de un concepto de arquitectura de software que puede ser implementado en cualquier tecnología de desarrollo de aplicaciones web. Por medio del desarrollo de esta plataforma se espera facilitar y mejorar la comunicación de los participantes en el negocio agrícola, proporcionando un medio de información global moderno que también fomente la exportación e importación de productos agrícolas.

La aplicación web desarrollada consta de 3 módulos:

1. Módulo de Usuarios
2. Módulo de Acciones Sociales y E-commerce
3. Módulo de Repositorio de Documentos

El primer módulo permite la creación, borrado, y demás operaciones CRUD de usuarios, perfiles, etcétera dentro del sistema. El segundo módulo permite la interacción social entre usuarios e interacción de negocios entre los mismos. Mientras que el tercer y último módulo les permite subir, borrar, actualizar y leer archivos relacionados al negocio agrícola, así como fotos, videos, y documentos.

La aplicación fue desarrollada usando la plataforma Java EE y el framework Spring que se despliega en un servidor o contenedor de aplicaciones Java que será accesible en Internet.

Palabras clave: REST API, Red Social, Negocio agrícola, Internet, Java.

Abstract

The difficult access to foreign and local agricultural markets and the lack of tools to facilitate communication between different participants in the agricultural business has turned into a problem that is plaguing this sector.

In addition, social networks have become means of access to information that keep us informed of any event today and let us know about the social activities of people. Everyday new platforms and services that provide different functionalities to its users arise.

This thesis proposes the introduction of a social platform specifically focused in agricultural business by developing a REST API. The REST API which is defined as a web application that consists of a set of web services to be consumed by an application. The advantage of the REST API is precisely that it is not tied to a specific technology, is a concept of software architecture that can be implemented in any web application technology. Through the development of this platform is expected to facilitate and improve communication of participants in the agricultural business.

The web application that has been developed consists of 3 modules:

1. Users Module
2. Social Actions and E-commerce Module
3. Documents Repository Module

The first module allows the creation, deletion, and other CRUD operations related to profiles and users. The second module enables social interaction between users and business interaction between them. While the third and final module allows them to upload, delete, update and read files related to agricultural business as well as photos, videos, and documents.

The application was developed using the Java EE platform and Spring framework that is deployed on a server or container for Java applications that will be accessible on the Internet.

Keywords: REST API, Social Network, Agricultural business, Internet, Java.

Introducción

Hoy en día existe una vasta cantidad de información que está al alcance de todos por medio del internet y que se nos facilita a través de buscadores como Google, hoy en día la humanidad está en contacto de manera permanente sin importar el lugar en el que se esté, tenemos incluso nuestros propios medios de difusión a través de las redes sociales como Facebook, Twitter, YouTube, etcétera. Es posible incluso darse cuenta del estado emocional de las personas por medio de las redes sociales. Hoy en día es muy difícil que los problemas que aquejan a la sociedad pasen desapercibidos.

Es obvio que las redes sociales han llegado para facilitar y ampliar la manera en que las personas se comunican y es precisamente el enfoque que esta Tesis busca, es decir, utilizar una plataforma de red social para facilitar la comunicación entre productores y comercializadores. Las redes sociales a diferencia de un Marketplace son plataformas que enganchan a los usuarios, porque los mantienen informados y les permiten conocer más sobre las personas con las cuales tienen alguna conexión o interacción, un Marketplace es solamente un sitio donde se oferta un producto con el fin de venderlo.

Para entender este documento en su totalidad se deben tener conocimientos de la Programación Orientada a Objetos, conocer el lenguaje de programación Java y conocimientos básicos sobre los diagramas de clase UML, conocimiento en conceptos de bases de datos, en general, tener conocimiento de ciertos términos empleados en el desarrollo de software.

Este documento se encuentra estructurado de la siguiente forma: El Capítulo 1 Planteamiento del problema muestra el planteamiento de la problemática actual del negocio agrícola para la cual fue desarrollada esta Tesis, los Capítulos 2 y 3 son capítulos introductorios. El Capítulo 2 La agricultura en el mundo habla sobre la agricultura y su crecimiento proyectado para los siguientes años así como los participantes del negocio agrícola, el Capítulo 3 Las redes Sociales describe la teoría de las redes sociales y su clasificación. El Capítulo 4 Tecnologías para el desarrollo de aplicaciones empresariales detalla las tecnologías de software utilizadas para el desarrollo de la aplicación. El Capítulo 5 Diseño de la aplicación habla sobre el modelo de la aplicación, flujo y modelado de los datos. El Capítulo 6 Resultados es el capítulo de resultados y muestra los resultados de las pruebas hechas al código desarrollado para este REST API. Finalmente está el Capítulo 7 Conclusiones y trabajo futuro que detalla las conclusiones a las que se llegó al término de esta Tesis y el trabajo pensado a futuro de este proyecto.

Capítulo 1 Planteamiento del problema

Una de las problemáticas actuales del negocio agrícola es la dificultad de acceso a mercados externos o incluso locales en los cuales sea posible promover, ofertar, vender y comprar productos agrícolas. El hecho de que sea una problemática no implica que no existan maneras de obtener acceso a diferentes mercados externos o locales, sino más bien la problemática reside en que esos medios de acceso no son medios accesibles o adecuados, de forma que, un medio óptimo es una manera sencilla, eficaz, barata, rápida y de una amplia proyección para dar a conocer un producto a diferentes mercados. El medio con mayor adopción para el acceso a mercados de diferentes países son las ferias internacionales.

Las ferias internacionales son grandes exposiciones comerciales de productos y servicios, organizados con el objetivo de facilitar el intercambio comercial entre países. Constituyen una instancia para promover productos y/o servicios, realizar contactos de negocios con personas de todas partes del mundo (o al menos de la región económica en que ésta se realiza). Las ferias comerciales son consideradas como una de las mejores oportunidades para reunirse con los clientes actuales y contactar a los clientes potenciales sin mucho esfuerzo, aprovechando la concentración en un mismo lugar y en un lapso reducido de tiempo de importadores, distribuidores, etc. Sin embargo una feria internacional implica una considerable inversión de recursos y tiempo, por lo general, de varios miles de dólares por lo que la participación en este tipo de eventos debe ser planeada cuidadosamente para asegurar la obtención de resultados deseados (ProMéxico, 2010).

El costo de acudir a una feria internacional es quizás un gasto que no todo productor agrícola pueda solventar y por lo tanto su principal desventaja, así también, la corta duración de la feria representa otro punto en contra. ¿De qué manera podría un productor agrícola realizar contactos de negocios con personas en todas partes del mundo y estar en contacto con ellos de manera permanente y al mismo tiempo promover sus productos con una proyección a nivel internacional sin la necesidad de gastar sumas importantes de dinero y sin la necesidad de abandonar su país, su localidad o región?

Una posible respuesta es, obviamente, por medio del internet. El internet es una estructura computacional global por medio de la cual la humanidad se encuentra conectada, comunicada e informada sobre los diferentes acontecimientos que se dan en los diferentes lugares del planeta de manera prácticamente instantánea y de forma permanente. El Internet es una plataforma ideal sobre la cual puede ser desarrollada una aplicación basada en un modelo que permita la interacción social entre usuarios y de la misma forma la interacción comercial entre ellos, es decir, una plataforma de red social. ¿Por qué no desarrollar una aplicación web basada en el modelo de las redes sociales y que esté enfocada completamente en el negocio agrícola? Que permita la comunicación permanente e instantánea entre los diferentes participantes del negocio

agrícola en cualquier parte del mundo, permitiéndoles entablar negociaciones, ofertar y promocionar sus productos a nivel internacional y al mismo tiempo ser una fuente de información que facilite en cierta medida los procesos de exportación de los productos agrícolas.

1.1 Objetivo

El objetivo de esta tesis es desarrollar un REST API para una plataforma de red social global enfocada en el negocio agrícola, tomando como entidades principales al productor, al comercializador y a los diferentes productos que sirven como géneros vendibles entre ellos.

1.2 Justificación

La plataforma sirve como medio para facilitar y agilizar la forma de llevar a cabo el negocio agrícola. Fomentando la relación social entre los participantes y por ende acercando a productores y comercializadores (exportadores e importadores), quienes son los beneficiados directos en el desarrollo de una aplicación como ésta.

La importancia en el desarrollo de herramientas que faciliten y mejoren la manera en cómo se lleva a cabo el negocio agrícola y que brinden información sobre los procesos de exportación de los productos agrícolas es importante puesto que facilitan los procesos que se llevan a cabo en el negocio agrícola lo cual resulta en una disminución de costos, de desinformación y de tiempo para las partes involucradas, lo cual representa un beneficio para las mismas.

En la actualidad la agricultura es una actividad humana en la cual se utilizan muchas aplicaciones científicas y tecnológicas principalmente del área de la ingeniería genética y la biotecnología en la modificación de cultivos con el fin de mejorarlos para lograr características que sean más benéficas para el ser humano, como por ejemplo aumentar el tamaño de los frutos y de las semillas, desarrollar en los mismos tolerancia a los pesticidas, a las sequías, entre otras. Es claro que la tecnología está presente en la agricultura, al menos en los procesos de cultivo y producción de los productos agrícolas y debe estarlo debido a que la agricultura es la columna vertebral de la economía de los países y abastece de alimentos a la población mundial.

Debido a la importancia que tiene la agricultura y las actividades económicas y sociales relacionadas a ella, es necesario que se den avances tecnológicos no solo en los procesos de cultivo sino también en las actividades socioeconómicas afines como lo es el negocio agrícola a manera de que estos avances contribuyan al desarrollo, fortalecimiento y crecimiento de las actividades agrícolas y a la subsistencia de la población relacionada con estas actividades.

1.3 Propuesta

La propuesta de esta Tesis es el desarrollo de un REST API para una plataforma de red social enfocada en el negocio agrícola, haciendo hincapié en que solo se ha desarrollado el API, es decir, el conjunto de funciones o métodos que pueden ser invocados por otra aplicación. El REST API debe proveer funcionalidades que permitan a una persona hacer lo siguiente:

1. Crear un usuario y proveer todas las funcionalidades de manejo de usuarios como actualización de datos, poder darse de baja de la aplicación, etc.
2. Poder dar de alta un perfil especializado en el negocio agrícola que para el alcance de tesis únicamente serán productor y comercializador y las operaciones para el adecuado manejo de la información de los perfiles.
3. Poder dar de alta los productos agrícolas con los que el usuario realice transacciones, especificando información que muestre a los demás usuarios que vean el producto las certificaciones con las que cuenta y demás detalles técnicos de interés para los usuarios y proveer las demás operaciones que permitan el manejo adecuado de la información de sus productos.
4. Interacción social con los demás usuarios de la aplicación, en el sentido de que un usuario pueda relacionarse con algún otro usuario por medio de alguna interacción provista por el API como el seguimiento entre usuarios, de forma que se esté informado de la actividad social de otro usuario.
5. Interacción de negocio con los demás usuarios de la aplicación, debido a que se trata de una plataforma enfocada específicamente en el negocio agrícola debe existir un tipo de interacción que permita a los usuarios expresar su interés en cierto producto o productos de otro usuario con el propósito de realizar negociaciones.

Las anteriores funcionalidades son las características básicas con las que debe de contar la aplicación para ser una herramienta útil para un productor o un comercializador y están descritas de manera breve y general, en los capítulos posteriores se describen a fondo todas las funcionalidades que brinda el API.

El desarrollo de una aplicación que use el API desarrollado va más allá del alcance de la tesis. Un conjunto de funcionalidades y otras características planeadas en el desarrollo futuro de este trabajo se encuentran listadas en el Capítulo 7.

Capítulo 2 La agricultura en el mundo

La agricultura es una de las actividades económicas más importantes y es esencial para la supervivencia, aseguramiento de la producción de alimento y obtención de ingresos. La agricultura es una actividad dependiente del ambiente que envuelve el uso del ecosistema y recursos ambientales como tierra, suelo, agua y energía. Es además la actividad humana que más volumen de agua consume en todo el mundo.

La agricultura es una actividad que ha ido creciendo, desarrollándose y mejorando sus procesos con el paso de los años. En la actualidad, la población del mundo supera los 6 mil millones de personas. Cada una de esas personas alcanzará una ingesta diaria aproximada de 2,700 kcal en promedio, cuando en el año de 1950 2,500 millones de personas disponían de menos de 2,450 kcal por persona. Esto significa que durante los últimos 50 años, el aumento de la producción agrícola mundial ha sido 1.6 veces superior a la producción total conseguida en 1950. Este extraordinario incremento de la producción de alimentos se explica en razón de:

- La difusión en los países desarrollados de la revolución agrícola moderna (caracterizada por la motorización, la mecanización en gran escala, la selección, la utilización de productos químicos y la especialización) y su expansión en algunos sectores de los países en desarrollo;
- La existencia, más notable en los países en desarrollo, de una revolución verde (caracterizada por la selección de determinadas variedades de cereales y otras plantas domésticas de alto rendimiento adecuadas a las regiones cálidas, y por la utilización de productos químicos), una forma de revolución agrícola moderna que no depende de una motorización mecanizada en gran escala;
- La expansión de la superficie de regadío, que ha pasado de 80 millones de hectáreas en el año de 1950 a unos 270 millones de hectáreas en la actualidad;
- La expansión de la superficie cultivable de la tierra bajo cultivos permanentes, que ha pasado en ese mismo período de 1,330 millones de hectáreas a 1,500 millones de hectáreas;
- La adopción de sistemas agrícolas mixtos que utilizan profusamente la biomasa disponible (combinando los cultivos, la arboricultura, la ganadería y, en ocasiones, la piscicultura) en la mayor parte de las zonas densamente pobladas del mundo que no disponen de nuevas tierras para la agricultura y para el riego.

En 1950, la agricultura empleaba a 700 millones de personas en todo el mundo y utilizaba menos de 7 millones de tractores (4 millones en los Estados Unidos, 180 mil en Alemania occidental y 150 mil en Francia) y menos de 1.5 millones de cosechadoras. Actualmente, los 1,300 millones de personas que se dedican a la agricultura disponen de 28 millones de tractores y 4.5 millones de cosechadoras, principalmente en los países desarrollados. En 1950 sólo se aplicaron 17

millones de toneladas de fertilizante mineral, cuatro veces más que en 1900 pero ocho veces menos que en la actualidad.

En 1950, los rendimientos de los cultivos eran de 1,000 kg/ha para el trigo, 1,500 kg/ha para el maíz, 1,600 kg/ha para el arroz y 1,100 kg/ha para la cebada, cifras prácticamente idénticas a las de comienzos de siglo. Desde entonces, los rendimientos se han duplicado o triplicado.

La revolución agrícola moderna que ha triunfado en los países desarrollados en la segunda mitad del siglo XX se ha basado en la aparición de nuevos medios de producción y comercio, que a su vez derivaron de las revoluciones acaecidas en la industria, la biotecnología, el transporte y las comunicaciones.

La revolución de las comunicaciones, que se basa, en parte, en la revolución del transporte, pero también en las telecomunicaciones, proporcionó los medios para el suministro de información y para las transacciones comerciales a larga distancia, que impulsaron el comercio distante y la organización de estructuras administrativas, productivas, financieras y comerciales en gran escala que son parte integrante de la revolución industrial y agrícola moderna (Organización de las Naciones Unidas para la Agricultura y la Alimentación, 2000).

2.1 El mercado agrícola

En esta sección se presentan las cifras de producción actuales de los cultivos de trigo, arroz (molido) y un conjunto de granos que en el idioma inglés se agrupan por el término “coarse grains” que son el maíz, el grano de sorgo y la soya, los datos son a nivel país y mundial. Además, se presenta una tabla con los valores de consumo de cereales estimados en los siguientes años, se muestran gráficas de proyección de aumento de la población mundial y por último se muestran gráficas con las proyecciones de importaciones y exportaciones en el periodo comprendido por los años 2013-2023 del maíz, arroz, trigo y soya. Lo anterior con el fin de mostrar cómo es la producción actual en el mundo de los productos agrícolas, cuál es el consumo actual y en el futuro y cómo se comportarán las exportaciones e importaciones de los productos agrícolas en el futuro debido al aumento de la población en el mundo.

Tabla 2.1 Producción de cultivos de trigo en millones de toneladas métricas.

	2012-2013	2013-2014	Proyección 2014-2015 Mayo	Proyección 2014-2015 Junio
Mundo	657.3	714	697	701.6
EUA	61.7	58	53.4	52.8

Canadá	27.2	37.5	28.5	28.5
México	3.2	3.4	3.9	3.9
Unión Europea	133.8	143.3	144.9	146.3
Rusia	37.3	52.1	52	53
Ucrania	15.8	22.3	20	20
China	121	121.7	123	124
India	94.9	93.5	94	95.9
Pakistán	23.3	24	24.5	24.5
Argentina	9.3	10.5	12.5	12.5
Brasil	4.4	5.3	6	6
Australia	22.5	27	25.5	25.5
Sudáfrica	1.9	1.8	1.8	1.8
Turquía	15.5	18	15	15
Todos los demás países	85.2	95.6	92.1	92

Fuente: Tabla 01 World Crop Production Summary, United States Department of Agriculture (USDA), Foreign Agricultural Service.

De la Tabla 2.1 puede apreciarse que los mayores productores de trigo en el periodo 2012-2013 son Estados Unidos con casi el 10 por ciento del total de trigo producido en el mundo, la Unión Europea con poco más del 20 por ciento y la India con casi el 15 por ciento, sumando entre estos países casi la mitad de la producción total mundial, en el periodo 2013-2014 la producción de trigo de los Estados Unidos disminuyó al igual que el de la India, caso contrario al de la Unión Europea que continuó y continuará en aumento en el futuro cercano.

Tabla 2.2 Producción de cultivos de granos (maíz, soya y grano de sorgo) en millones de toneladas métricas.

	2012-2013	2013-2014	Proyección 2014-2015 Mayo	Proyección 2014-2015 Junio
Mundo	1,318.1	1,272.6	1,257.2	1,258.7
EUA	286	369.4	368.6	368.6

Canadá	24.4	28.7	23.1	23.1
México	28.9	29.8	30.1	30.1
Unión Europea	146.1	158.3	152.2	153.1
Rusia	28.7	35.7	37.5	38
Ucrania	29.5	39.9	34.3	35.3
China	212.2	224.3	226.6	226.6
India	39.9	42.7	40.7	41.7
Indonesia	8.5	9.1	9.2	9.2
Pakistán	5.6	5.6	5.6	5.6
Tailandia	4.7	6	5	5
Argentina	37.2	35.2	35.2	35.2
Brasil	84.3	79.1	77.1	77.1
Australia	11.2	12.4	11.8	11.8
Sudáfrica	12.9	15.1	14	14
Turquía	10.6	13.1	11.4	9.6
Todos los demás países	167.5	170.7	174.8	174.7

Fuente: Tabla 01 World Crop Production Summary, United States Department of Agriculture (USDA), Foreign Agricultural Service.

En el caso de la producción de maíz, grano de sorgo y soya, de la Tabla 2.2, se pueden observar los 3 productores más grandes que son Estados Unidos, China y la Unión Europea. Para el caso de los Estados Unidos los niveles de producción del periodo 2013-2014 se elevaron significativamente en comparación a los niveles del periodo 2012-2013, en el caso de China los niveles de producción seguirán en aumento. Para el caso de la Unión Europea la producción aumentó del periodo 2012-2013 al periodo 2013-2014 pero tenderá a disminuir de acuerdo a las proyecciones que corresponden al periodo 2014-2015.

Tabla 2.3 Producción de cultivos de arroz, molido, en millones de toneladas métricas.

	2012-2013	2013-2014	Proyección 2014-2015	Proyección 2014-2015

			Mayo	Junio
Mundo	471.6	477.5	480.7	480.7
EUA	6.3	6.1	6.8	6.8
México	0.1	0.1	0.1	0.1
Unión Europea	2.1	1.9	2	2
Rusia	0.7	0.6	0.7	0.7
Ucrania	0.1	0.1	0.1	0.1
China	143	142.3	144	144
India	105.2	106.3	106	106
Indonesia	36.6	37.4	37.7	37.7
Pakistán	5.8	6.6	6.7	6.7
Tailandia	20.2	20.5	20.5	20.5
Argentina	1	1	1	1
Brasil	8	8.6	8.5	8.5
Australia	0.8	0.7	0.7	0.7
Turquía	0.5	0.5	0.5	0.5
Todos los demás	141.1	144.8	145.5	145.5

Fuente: Tabla 01 World Crop Production Summary, United States Department of Agriculture (USDA), Foreign Agricultural Service.

El término arroz molido se refiere al arroz blanco, los dos productores más grandes de arroz blanco son China e India con poco más de la mitad de la producción de arroz blanco entre ambos países y de acuerdo a la Tabla 2.3 seguirán aumentando sus niveles de producción en comparación al periodo 2012-2013.

Tabla 2.4 Producción de cultivo total de granos incluyendo arroz (molido), trigo, maíz, grano de sorgo y soya, en millones de toneladas métricas.

	2012-2013	2013-2014	Proyección 2014-2015	Proyección 2014-2015

			Mayo	Junio
Mundo	2,267.0	2,464.0	2,434.9	2,441.0
EUA	354.0	433.5	428.8	428.2
Canadá	51.6	66.2	51.6	51.6
México	32.2	33.3	34.1	34.1
Unión Europea	282.0	303.6	299.1	301.3
Rusia	67.1	88.4	90.2	91.7
Ucrania	45.4	62.3	54.4	55.4
China	476.2	488.3	493.6	494.6
India	240.1	242.5	240.7	243.6
Indonesia	45.1	46.5	46.9	46.9
Pakistán	34.7	36.2	36.8	36.8
Tailandia	24.9	25.5	25.5	25.5
Argentina	47.6	45.0	48.7	48.7
Brasil	96.6	93.0	91.6	91.6
Australia	34.5	40.1	38.0	38.0
Sudáfrica	14.7	16.9	15.8	15.8
Turquía	26.6	31.6	26.9	25.1
Todos los demás	393.8	411.1	412.3	412.2

Fuente: Tabla 01 World Crop Production Summary, United States Department of Agriculture (USDA), Foreign Agricultural Service.

La Tabla 2.4 muestra que los productores mayores de granos (arroz blanco, maíz, soya, grano de sorgo y trigo) son Estados Unidos, China, la Unión Europea e India y seguirán aumentando su producción de granos al menos en los periodos especificados que representan a los periodos actuales (2012-2015). La Tabla 2.4 muestra también que la producción mundial en el periodo 2013-2014 aumentó en comparación al periodo anterior y se proyecta que la producción bajará un poco en el siguiente periodo pero no a los niveles del periodo 2012-2013.

El consumo global de cereales a nivel mundial proyectado en el periodo comprendido por los años 2013-2022 se muestra en la Tabla 2.5.

Tabla 2.5 Proyección de consumo mundial de cereales en el periodo 2013-2022 en miles de toneladas.

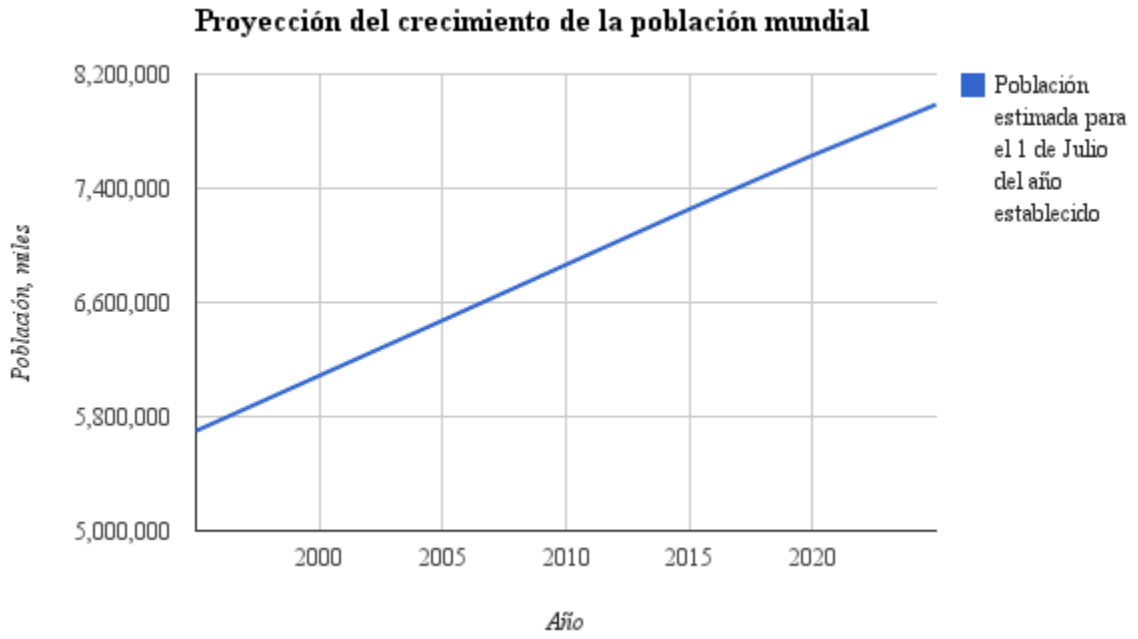
Año	Trigo	Coarse grains	Arroz
2013	692,224.5	1,203,903.7	486,688.2
2014	700,774.8	1,231,697.7	496,060.7
2015	712,004.8	1,247,310.7	505,740.8
2016	723,751.6	1,267,641.2	510,359.9
2017	733,070.6	1,289,564.2	517,233.9
2018	742,948.6	1,314,146.1	524,446.2
2019	753,590.5	1,338,035.7	531,843.8
2020	763,367.4	1,361,121.9	538,739.7
2021	773,155.9	1,383,999	545,188.1
2022	782,429.5	1,408,174.5	551,294.7

Fuente: (OECD-FAO, 2014).

Se proyecta que la demanda global de productos agrícolas continuará creciendo durante el periodo de proyección comprendido por los años 2014-2023. Al mismo tiempo, se tiene proyectado que la producción mundial de productos agrícolas va a incrementar más rápidamente que la población mundial, permitiendo un pequeño incremento en el uso per cápita mundial promedio de la mayoría de los productos agrícolas (Interagency Agricultural Projections Committee, 2014).

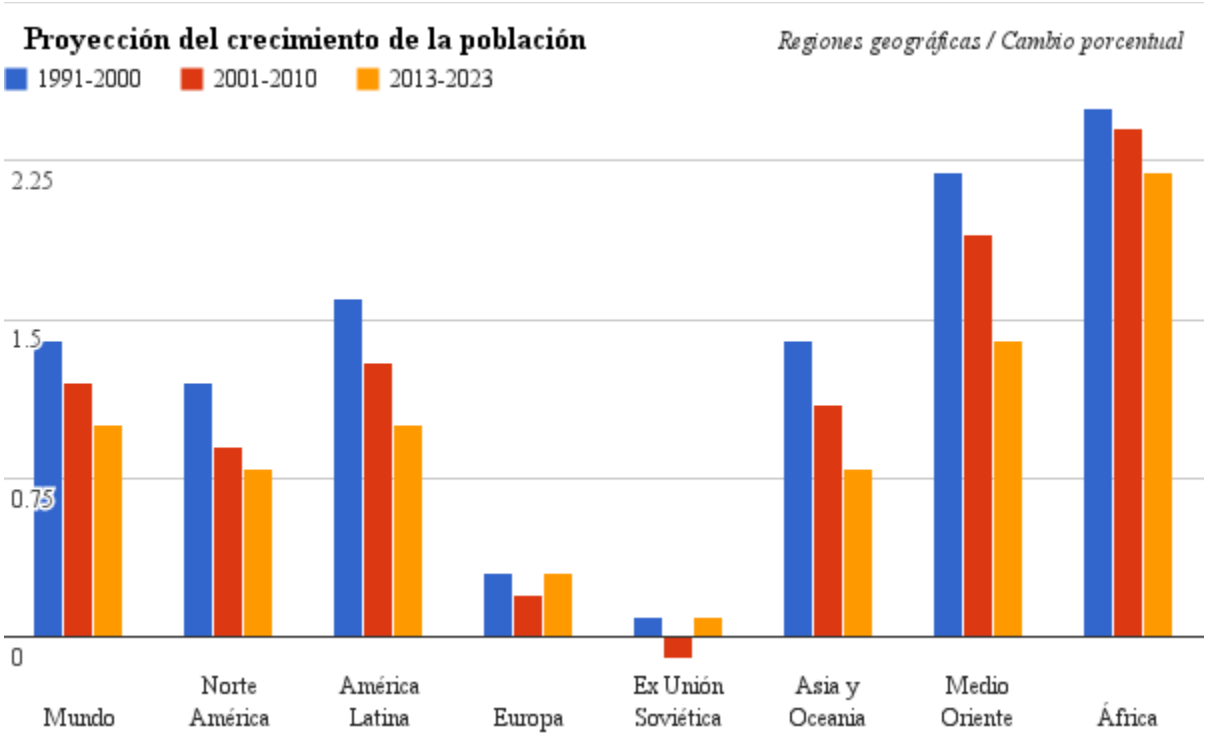
Se espera que la población mundial crezca considerablemente en los próximos años, aunque en una tasa más lenta que en el pasado, y con considerables diferencias entre regiones. En las siguientes cuatro décadas, la población mundial está pronosticada a incrementar en 2 mil millones de personas para exceder los 9 mil millones de personas para el año 2050. Estimaciones de la FAO indican que para satisfacer la demanda global, la producción agrícola tendrá que incrementar un 60 por ciento de su niveles del periodo 2005-2007 (Food and Agriculture Organization of the United Nations, 2013).

Figura 2.1 Gráfica de la proyección del crecimiento de la población mundial.



Fuente: (U.S. Department of Commerce, U.S. Census Bureau, 2013).

Figura 2.2 Gráfica de la proyección del crecimiento de la población, cambio porcentual en el crecimiento de la población mundial.



Fuente: (Interagency Agricultural Projections Committee, 2014).

La agricultura (incluyendo cultivos, ganado, bosques, pesca y acuacultura) es la principal actividad humana responsable del manejo de recursos naturales a niveles locales y regionales. Treinta por ciento de la tierra es usada para cultivos y pasturas, y setenta por ciento de toda el agua dulce extraída está dirigida directamente hacia la irrigación para la producción de comida que la gente y el ganado necesitan para un suministro estable de comida.

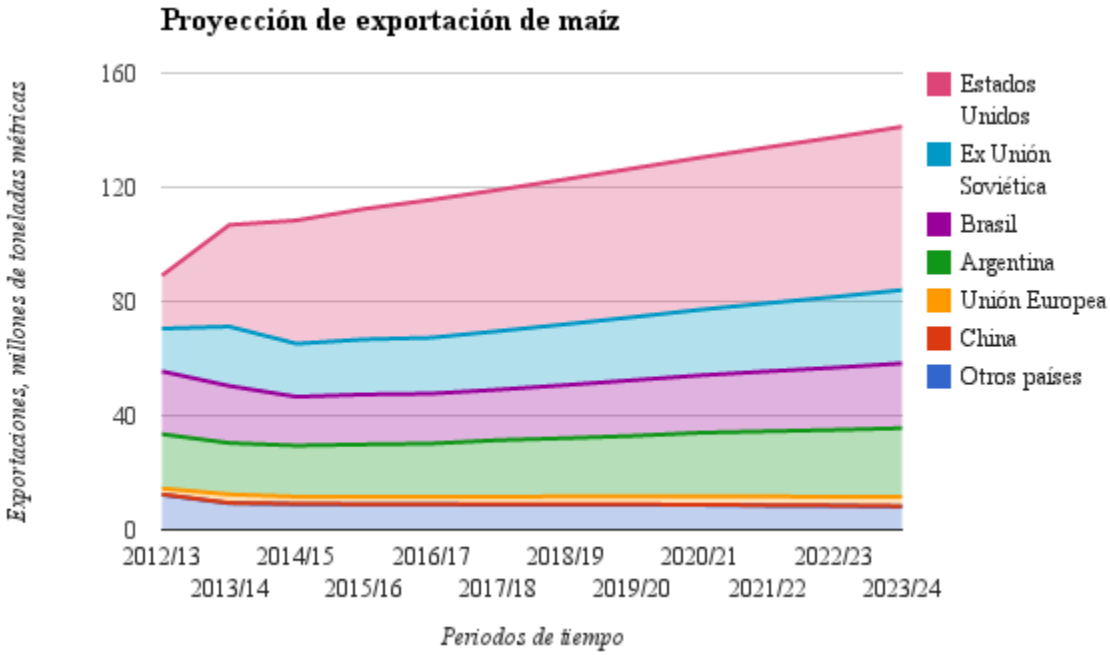
Nuevas y tradicionales demandas de productos aumentan la presión sobre los escasos recursos agrícolas. Mientras que el sector agrícola será forzado a competir por tierra y agua con la expansión de asentamientos urbanos y zonas industriales, también será requerido satisfacer el crecimiento en la demanda de la "biobased economy" (economía bio-basada), cada vez más por medio de la bioenergía y nuevos mercados emergentes para productos industriales renovables y sustentables.

Aunque la agricultura continuará siendo un usuario mayor de tierra y agua, necesitará buscar nuevas maneras de mantener esos recursos para seguir siendo viable, y para minimizar impactos negativos sobre los ecosistemas y el bienestar humano. Asegurar comida y agua adecuada para todos mientras se logre un desarrollo rural sustentable que gire en torno a una renovada administración para el manejo responsable de los recursos naturales, y por lo tanto sobre un sistema agrícola sostenible (Food and Agriculture Organization of the United Nations, 2013).

Debido al incremento de la población mundial, serán requeridos más productos agrícolas que satisfagan la demanda de la población que con el paso de los años seguirá aumentando.

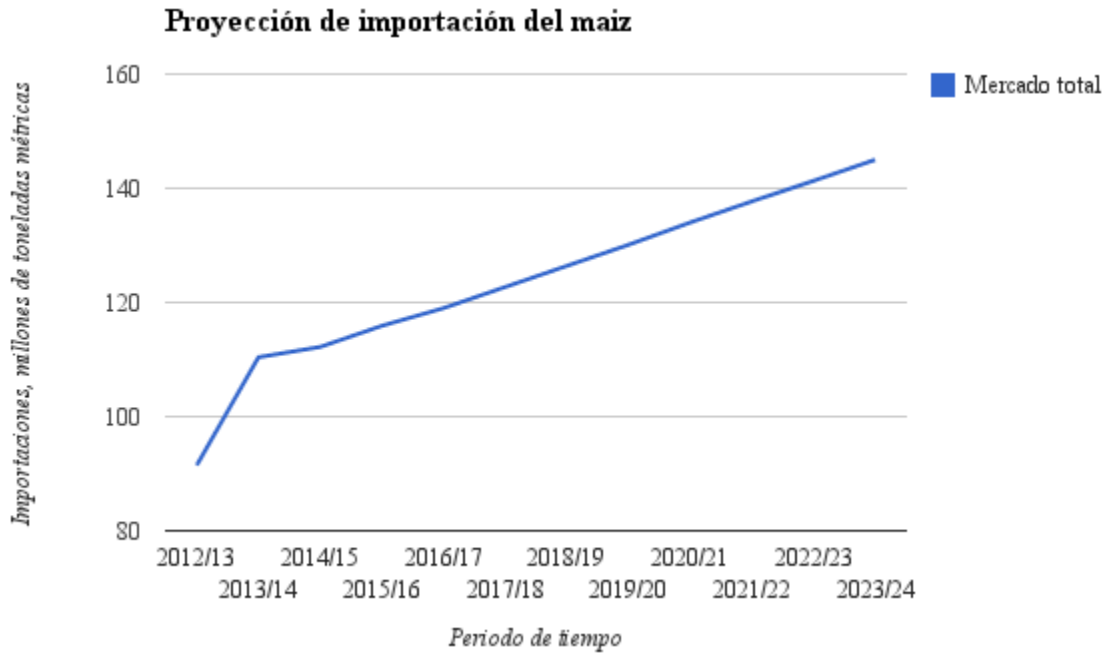
Las siguientes gráficas muestran las proyecciones de importaciones y exportaciones a nivel mundial de diferentes productos agrícolas.

Figura 2.3 Gráfica de la proyección del número de exportaciones de maíz de diferentes países.



Fuente: (Interagency Agricultural Projections Committee, 2014).

Figura 2.4 Gráfica de la proyección del número de importaciones de maíz a nivel mundial.

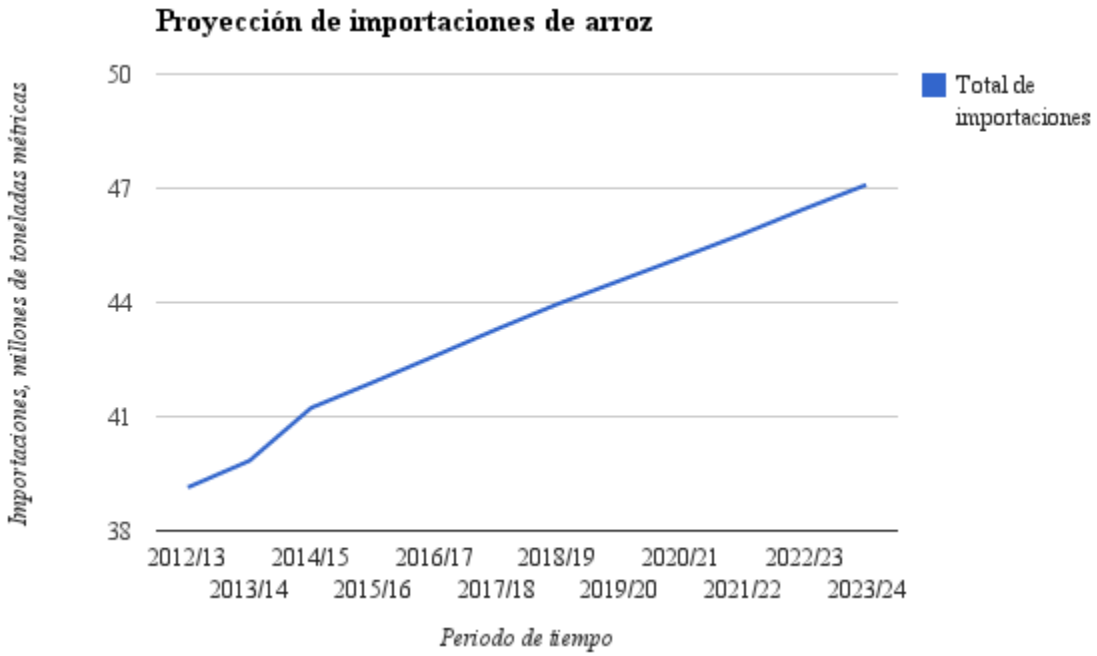


Fuente: (Interagency Agricultural Projections Committee, 2014).

De acuerdo a la

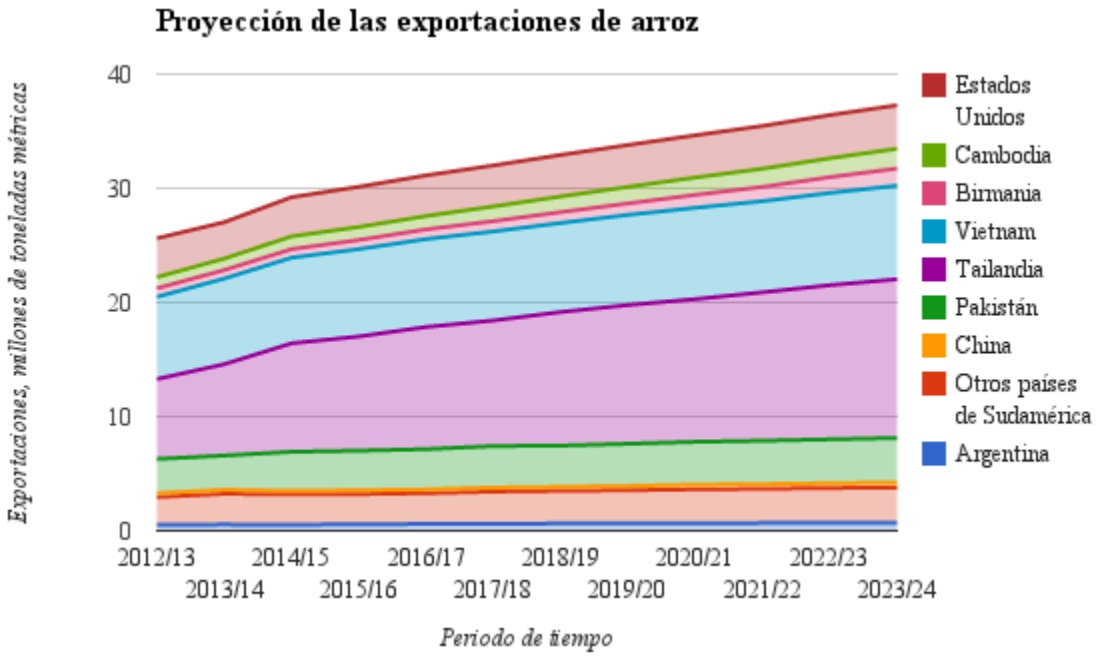
Figura 2.3 y a la Figura 2.4 la tendencia en el número de importaciones y exportaciones de maíz es que éstas seguirán incrementando en los siguientes años.

Figura 2.5 Gráfica de la proyección del número total de importaciones de arroz a nivel mundial.



Fuente: (Interagency Agricultural Projections Committee, 2014).

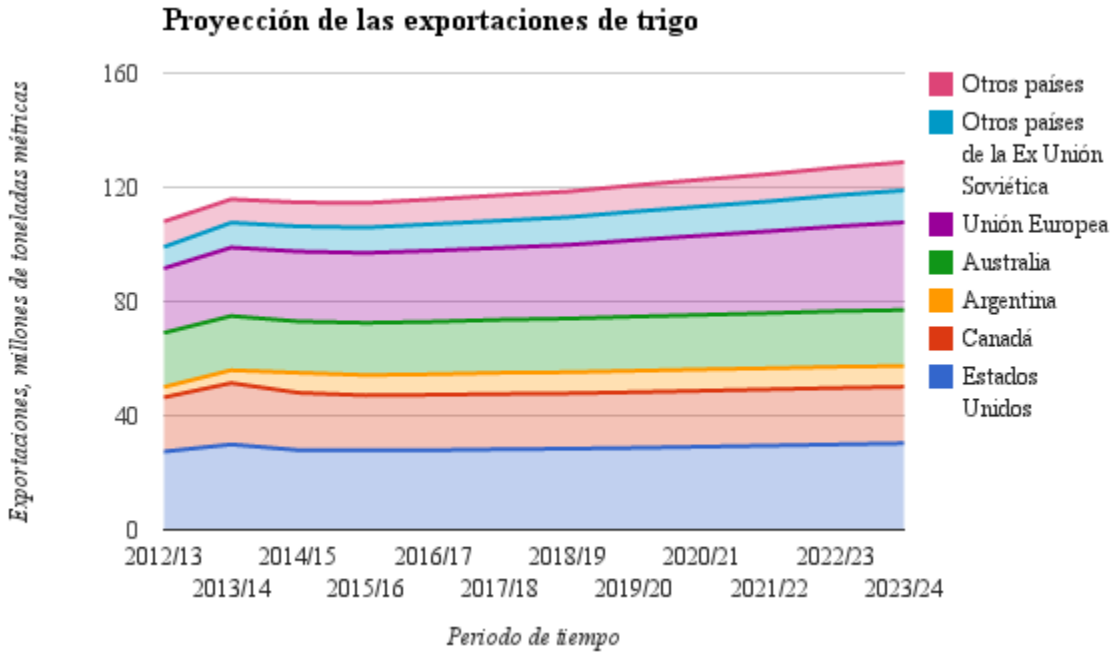
Figura 2.6 Gráfica de la proyección de exportaciones de arroz de diferentes países.



Fuente: (Interagency Agricultural Projections Committee, 2014).

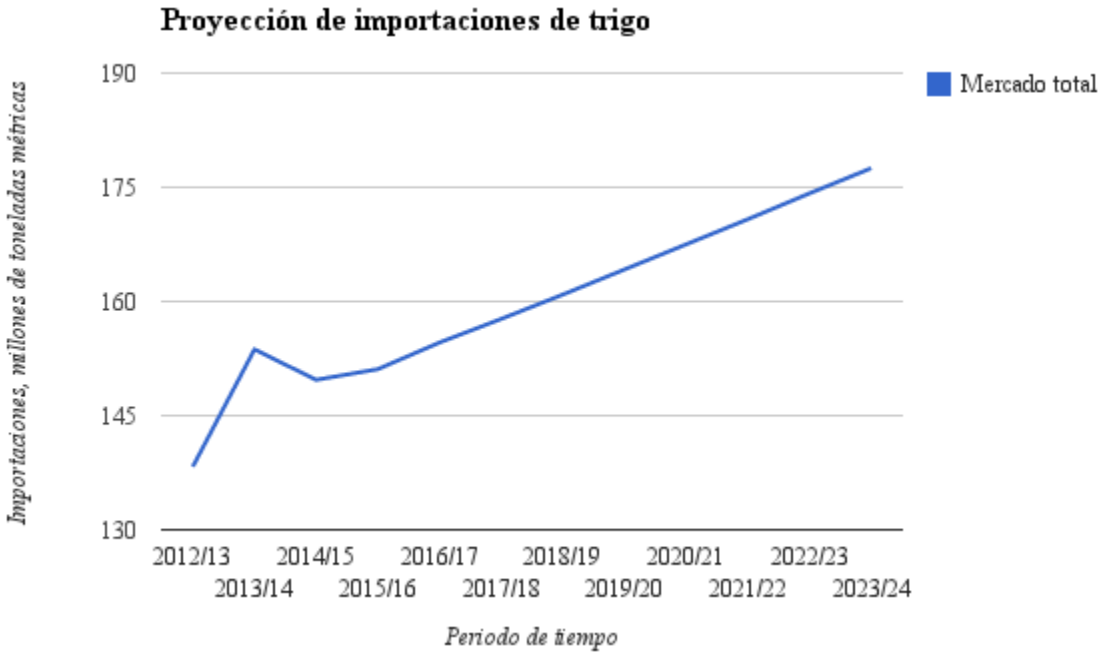
De acuerdo a la Figura 2.5 y a la Figura 2.6 el número de exportaciones de arroz en los siguientes años seguirá en aumento en los próximos años.

Figura 2.7 Gráfica de proyección del número de exportaciones de trigo de diferentes países.



Fuente: (Interagency Agricultural Projections Committee, 2014).

Figura 2.8 Gráfica de la proyección del número total de importaciones de trigo a nivel mundial.



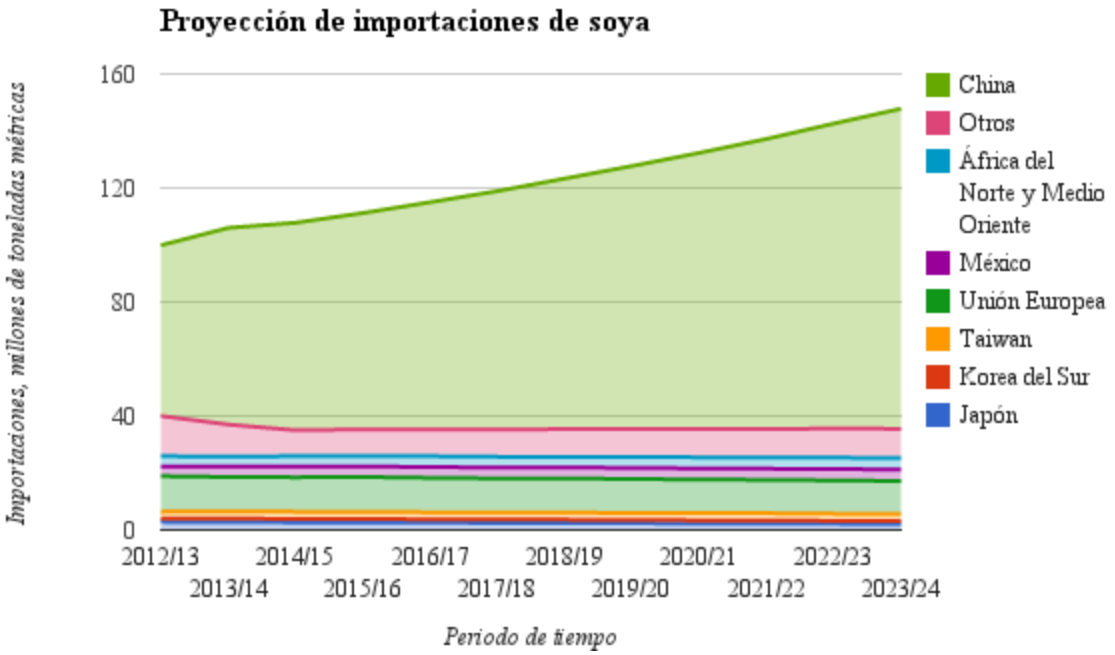
Fuente: (Interagency Agricultural Projections Committee, 2014).

Las dos gráficas correspondientes a la Figura 2.7 y a la Figura 2.8 muestran el crecimiento en el número de exportaciones e importaciones de trigo en los siguientes años.

Figura 2.9 Gráfica de la proyección del número de exportaciones de soya a nivel mundial.

Fuente: (Interagency Agricultural Projections Committee, 2014).

Figura 2.10 Gráfica de la proyección del número de importaciones de soya de diferentes países.



Fuente: (Interagency Agricultural Projections Committee, 2014).

De acuerdo a la Figura 2.9 y a la Figura 2.10 el número de importaciones y exportaciones de soya seguirá en aumento en los años venideros.

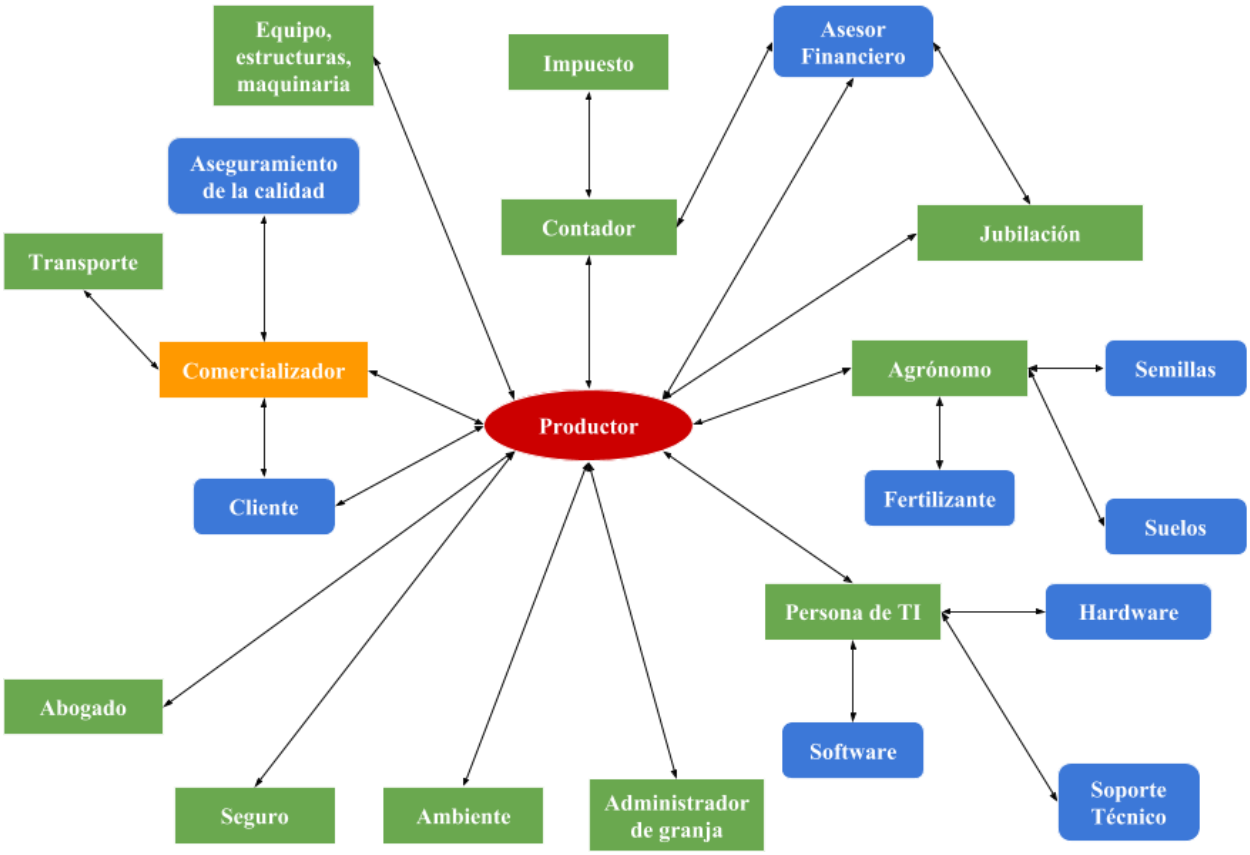
Aunado a lo anterior y a que no todos los países cuentan con la suficiente extensión de tierra o no cuentan con la tecnología necesaria para autoabastecerse tendrán que elevar el número de sus importaciones para poder satisfacer la demanda de sus naciones.

Como puede apreciarse en las proyecciones anteriores, la tendencia en exportaciones e importaciones en los productos agrícolas básicos es que seguirán aumentando en los años siguientes. Puesto que la demanda de estos productos irá en aumento con el paso de los años como se mencionó anteriormente. Por lo tanto, el mercado agrícola seguirá creciendo.

2.2 Participantes del negocio agrícola

El negocio agrícola es un sistema complejo que cuenta con una gran variedad de participantes que intervienen en él. La Figura 2.11 muestra un enfoque en el cual el productor es el centro de las relaciones entre los diversos participantes en el negocio.

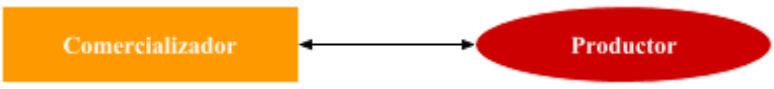
Figura 2.11 Participantes del negocio agrícola teniendo como punto central de las diferentes relaciones el rol de productor.



Fuente: (Stone, 2005).

Para el alcance de este proyecto de tesis no se toma en cuenta la totalidad de los diferentes participantes mostrados en la Figura 2.11, sino solo una pequeña parte que representa las interacciones entre el productor y comercializador como lo muestra la Figura 2.12. Dichas interacciones serán convertidas a un modelo computacional, sobre el cual será construido un REST API del cual se hablará más adelante en este documento.

Figura 2.12 Interacción entre comercializador y productor cubierta por el alcance de proyecto de tesis.



De acuerdo a la FAO la definición de productor es la siguiente:

"El productor es una persona civil o jurídica que adopta las principales decisiones acerca de la utilización de los recursos disponibles y ejerce el control administrativo sobre las operaciones de la explotación agropecuaria. El productor tiene la responsabilidad técnica y económica de la explotación, y puede ejercer todas las funciones directamente o bien delegar las relativas a la gestión cotidiana a un gerente contratado", (Pedrero, 2001).

Por lo tanto un productor es quien explota la tierra y produce los diferentes productos agrícolas que derivan de ella. Mientras que por su parte un comercializador es alguien que vende o compra productos para un cierto mercado, en este caso el mercado agrícola o alguno relacionado. Son precisamente estas interacciones de compra-venta entre productor y comercializador en las que se enfoca este proyecto de tesis.

Este capítulo es la justificación del por qué son necesarias nuevas herramientas en el negocio agrícola que faciliten la comunicación entre productores y comercializadores. Como se mostró en las secciones de este capítulo el aumento en la población traerá consigo un aumento en la demanda de productos agrícolas, lo que implica que, los productores deberán producir un mayor número de productos agrícolas, se deberán cultivar mayores extensiones de tierra, así como también, mejorar los procesos de cultivo para el incremento de la producción. La carencia de medios de acceso a mercados que sean baratos y accesibles debe ser de algún modo cubierta con el desarrollo de nuevas herramientas tecnológicas. El uso de una red social sería una herramienta ideal para esta problemática.

El siguiente capítulo es un capítulo dedicado enteramente a las redes sociales mostrando cuál es su definición, una breve historia de las plataformas de redes sociales que han existido y existen actualmente además de la clasificación de las plataformas de red social y sus principales características.

Capítulo 3 Las redes sociales

Las redes sociales hoy en día son plataformas que brindan no solo conexión entre amigos, familiares, contactos de trabajo, etcétera, sino que, ahora son medios de información que nos dan una idea de lo que está sucediendo a nuestro alrededor y en todo el mundo. A tal grado que un gran porcentaje de la población en el mundo utiliza estas plataformas para estar en contacto y para estar al día de lo que pasa en su entorno, pudiendo incluso decir que las redes sociales son parte de nuestra vida cotidiana, parte de nuestro estilo de vida. Es tanta la aceptación de tales plataformas que hoy en día se encuentran dentro de los primeros sitios más visitados en el mundo de acuerdo al portal Alexa (Alexa Internet, Inc., 2014).

Hoy en día muchas compañías multinacionales de toda índole cuentan con redes sociales internas, en las que fomentan la interacción entre sus empleados, proporcionando además un canal de comunicación hacia los departamentos administrativos y ejecutivos de las empresas. Ejemplos de estas redes sociales internas son: Ultimatix (Tata Consultancy Services, 2014) de Tata Consultancy Services y Staples Associate Connection (People Soft, 2006) de Staples Inc.

3.1 Definición

Una red social es una plataforma que brinda servicios basados en web que permiten a los individuos construir un perfil semipúblico dentro de un sistema delimitado, tener una lista de otros usuarios con los cuales se comparte una conexión, ver y navegar su lista de conexiones y la de otros usuarios dentro del sistema. La naturaleza y nombres de esas conexiones varían dependiendo de la plataforma de red social.

Lo que hace a las redes sociales únicas no es que permitan a las personas conocer nuevas personas, sino que habilitan a los usuarios para expresar y hacer visibles sus opiniones y estados de ánimo al igual que sus interacciones sociales con los demás usuarios.

El elemento central de una red social es el perfil, una vez que una persona ha decidido unirse a alguna plataforma de red social lo primero que se le pide es llenar una forma que contiene una serie de preguntas. El perfil se genera utilizando las respuestas a las preguntas que el nuevo usuario respondió y que típicamente son preguntas sobre información personal del usuario como edad, intereses, etcétera. La visibilidad del perfil depende de cada plataforma de red social y de la discreción del usuario.

Después de que una persona se ha registrado como usuario de una red social, se le pide que identifique a otros usuarios con los que tiene relaciones (amigos, conocidos, contactos, pareja, familia, entre otros). La manera en la que son llamadas estas relaciones difiere de cada

plataforma, en algunas plataformas como Facebook por ejemplo se les llama “amigos” en el caso de Twitter se les llama “seguidores”.

La exhibición pública de las conexiones o relaciones de los usuarios con otros usuarios es un componente crucial de las redes sociales. La lista de conexiones contiene enlaces a cada uno de los perfiles de los amigos o contactos en la lista, habilitando de esta manera que aquellos que estén viendo la lista puedan explorar toda la red de conexiones del usuario con otros usuarios solo al dar clic a cada uno de los enlaces contenidos en la lista. La visibilidad de la red de conexiones del usuario varía dependiendo de cada plataforma de red social al igual que la discreción del usuario.

Las plataformas de red social incluyen a su vez más funcionalidades como el envío de mensajes “comentarios” en los perfiles de usuario, algunas plataformas permiten el envío de mensajes privados entre usuarios “chats”. Más allá de todas estas funcionalidades, las diversas plataformas de red social varían en gran medida entre sí, puesto que algunas están más enfocadas en ciertas características que las distinguen de las demás como por ejemplo la compartición de fotos o videos; otras han sido incorporadas en blogs y en tecnologías de mensajería instantánea. Las hay enfocadas principalmente en dispositivos móviles y en web. Algunas plataformas están destinadas a solo cierto tipo de usuarios, por ejemplo, usuarios que habitan en regiones geográficas o lugares específicos, con ciertas características culturales, étnicas, religiosas, o de orientación sexual o política, entre muchas otras más (Boyd & Ellison, 2008).

3.2 Breve historia de las redes sociales

De acuerdo a la definición anterior, la primera plataforma de red social fue lanzada en el año de 1997. SixDegrees.com es una de las primeras manifestaciones del formato de las redes sociales de hoy en día. SixDegrees.com permitía a los usuarios la creación de perfiles, listar los amigos o conexiones del usuario y más delante permitía navegar las listas de conexiones entre usuarios. Aunque SixDegrees.com no fue el primer sitio en implementar las características anteriores, sí fue el primer sitio que las combinó.

De 1997 a 2001 un gran número de plataformas comenzaron soportando varias combinaciones de perfiles y de listas de “amigos” expresadas de manera pública. Diversas plataformas permitieron a los usuarios la creación de perfiles personales, profesionales, de citas. Los usuarios podían identificar “amigos” por medio de sus perfiles sin buscar la aceptación de esas conexiones.

En el año de 1999 fue lanzada la plataforma LiveJournal, la cual implementaba conexiones unidireccionales en las páginas de usuarios. Las personas marcaban a otras personas como “amigos” para seguir sus “journals” y manejar sus configuraciones de privacidad.

En el año 2001 surge una nueva ola de plataformas sociales entre las cuales están LinkedIn, Ryze, Tribe.net, Friendster, dentro de las cuales la más destacada hoy en día es LinkedIn que se ha convertido en un poderoso servicio empresarial. Para el año 2002 Friendster se lanza como complemento social a la plataforma Ryze, estaba destinado a establecer nuevas relaciones entre personas particularmente entre "amigos de amigos". Friendster tuvo un éxito significativo pero presentaba dificultades técnicas para poder ir a la par con su crecimiento y debido a ciertas restricciones a los usuarios y que muchos de éstos ingresaban información no confiable en el sitio, comenzó a perder popularidad en los Estados Unidos. Sin embargo, hubo cierta región en el continente asiático donde ganó mucha popularidad.

En el año 2003 surge MySpace que atraería la atención principalmente de algunos grupos de personas como adolescentes, bandas musicales y personas interesadas en dichas bandas. MySpace permitía a menores de edad utilizar sus servicios. Tuvo algunos problemas puesto que comenzaron a surgir escándalos debido a interacciones sexuales entre adultos y menores de edad. Una de las características principales de MySpace es que permitía a los usuarios personalizar sus páginas, puesto que no restringía la adición de HTML, de manera que, los usuarios podían definir fondos únicos y disponer de varios elementos de diseño.

En el año 2004 Mark Zuckerberg crea Facebook. Al inicio Facebook solo era accesible para alumnos de Harvard, en el año 2005 comienza a expandirse hasta eventualmente ser accesible a todo el mundo. Una de las características que diferencia a Facebook de las demás plataformas es que Facebook permite el desarrollo de aplicaciones por parte de desarrolladores externos que permiten a los usuarios de Facebook interactuar con ellas.

En 2006 surge Twitter, el concepto de esta plataforma de red social es el "microblogging" que se basa en el envío y lectura de mensajes de texto cortos llamados "tweets" originalmente la plataforma estaba planeada como una plataforma de comunicación basada en SMS (Short Message Service) de allí el por qué los caracteres están limitados a un número de 140. Eventualmente la plataforma ha sido bien recibida hasta llegar a ser una de las plataformas sociales más usadas en la actualidad.

En el año 2011 la compañía Google lanza una plataforma de red social con el nombre de Google+. Quizás la característica que distingue y diferencia a Google+ de las demás redes sociales es el hecho de que además de ser una plataforma de red social es un servicio de identidad, de manera que, Google+ asocia el perfil de los usuarios con el contenido web que han producido, ésto por medio de la integración de otros servicios que la compañía desarrolla como el servicio de correo electrónico Gmail, el sitio web para subir y compartir videos YouTube y Blogger el cual es un servicio para escribir y publicar blogs. Actualmente Google+ es una de las plataformas con mayor número de usuarios en el mundo.

El auge de las redes sociales indica un cambio en la organización de las comunidades en línea. Las redes sociales están organizadas alrededor de las personas, no los intereses. Las redes sociales están estructuradas como redes personales, con el individuo al centro de su propia comunidad. Ésto refleja estructuras sociales no mediadas, donde el mundo es compuesto de redes, no de grupos (Boyd & Ellison, 2008).

3.3 Clasificación de las redes sociales

Muchos de los estudios realizados sobre redes sociales para determinar su comportamiento se han enfocado únicamente en plataformas de redes sociales a gran escala, así que no es posible decir que los resultados de dichos estudios aplican a características generales o especiales de este tipo de plataformas.

Existen distintas clasificaciones que pueden darse a las diferentes plataformas de redes sociales en base a su estructura y en sus patrones de comunicación, estas clasificaciones se obtienen en base al análisis de una serie de índices llamados "network indexes" (índices de red). Las clasificaciones mostradas en las posteriores secciones y parte de su contenido se basan en la información obtenida y los procedimientos realizados en el artículo (Toriumi, y otros, 2011) en el cual se analizan un total de 615 plataformas de red social, cada una con más de 50 usuarios.

En base al análisis de los network indexes se puede deducir el comportamiento de las interacciones que se dan en las redes sociales como se muestra a continuación.

3.3.1 Clasificación en base a Network Indexes

Redes Small World (Redes de mundo pequeño)

Las redes "small world" se basan en el fenómeno small world. El principio que establece que muchos de nosotros estamos enlazados por cadenas cortas de conocidos (personas). Fue investigado primeramente como una cuestión de la sociología y es una característica de las redes creadas en la naturaleza y la tecnología (Kleinberg, 2000).

La manera más simple de formular el problema del small world es: Empezando con dos personas en cualquier parte del mundo, ¿Cuál es la probabilidad de que ellos se conozcan? Una formulación del problema un poco más sofisticada, toma en cuenta el hecho de que mientras las personas *X* y *Z* puedan no conocerse directamente, ellos podrían compartir un conocido (persona) en común quien conoce a ambos. Se puede pensar en una cadena de conocidos con la persona *X* conociendo a la persona *Y* y la persona *Y* conociendo a la persona *Z*. Además, uno puede

imaginar circunstancias en las cuales X está enlazado a Z no solo por un solo enlace sino por una serie de ellos, X, a, b, c, d,..., y, Z, eso es decir que una persona X conoce a una persona a que a su vez conoce a una persona b que a su vez conoce a una persona d y así consecutivamente hasta llegar a una persona que conoce a la persona Z.

Por lo tanto, una pregunta puede ser formulada: Dadas dos personas en cualquier parte del mundo ¿Cuántos enlaces intermedios entre conocidos son necesarios antes de que X y Z estén conectados? (Milgram S. , 1967).

Se le llamó a este concepto "Six degrees of separation" (Seis grados de separación), se asume que grados de separación es lo mismo que "distancia menos uno", donde la "distancia" es la longitud del camino (el número de arcos o saltos en el camino), concepto concebido originalmente por Frigyes Karinthy, en su historia corta de 1929 "Láncszemek" (Cadenas) que sugería que dos personas cualesquiera están distanciadas a lo mucho por seis enlaces de amistad.

De acuerdo al estudio llevado a cabo en el artículo "The Small-World Problem" escrito por el psicólogo social Stanley Milgram en 1967 el promedio del número de personas necesarias para enlazar a dos personas geográficamente separadas y elegidas al azar son 5 teniendo como número total de cadenas terminadas 44 (Milgram S. , 1967). En el año de 1969 en el artículo "An Experimental Study of the Small World Problem" escrito por Stanley Milgram y Jeffrey Travers se obtiene una media de 5.2 intermediarios con 64 cadenas terminadas variando entre 4.4 y 5.7 intermediarios dependiendo del número de muestras tomadas (Milgram & Travers, 1969).

Recientemente se han realizado diversos estudios sobre el concepto de seis grados de separación en las plataformas de redes sociales con mayor número de usuarios en el mundo como Facebook, véase (Backstrom, Boldi, Rosa, Ugander, & Vigna, 2012). De acuerdo a este artículo, para el caso de Facebook en el año 2012 la distancia promedio observada era de 4.74, correspondiendo a 3.74 intermediarios o grados de separación usando la red de usuarios activos de Facebook en su totalidad ($\approx$ 721 millones de usuarios, 69 mil millones de enlaces de amistad cuando fue escrito dicho artículo, en la actualidad al momento de escribir esta Tesis el número de usuarios activos es: 1,310 millones alrededor del mundo).

Una característica de las redes small world es que siempre hay un camino muy corto entre dos nodos cualesquiera (Kleinberg, 2000).

Redes Scale Free (Redes libres de escala)

Las redes "scale free" poseen una propiedad importante: algunos nodos tienen un número tremendo de conexiones a otros nodos, mientras que la mayoría de los demás nodos que conforman la red tienen solo un puñado de estas conexiones a otros nodos. Los nodos con una

gran cantidad de conexiones, llamados “hubs”, pueden tener cientos, miles o incluso millones de enlaces. En este sentido, la red parece no tener una escala.

Las redes scale free tienen además características importantes. Ellas son, robustas contra fallas accidentales pero vulnerables a ataques coordinados. El entendimiento de tales características podría llevar al desarrollo de nuevas aplicaciones en muchas áreas. Por ejemplo, en el área de la computación podría ser posible desarrollar estrategias para prevenir que los virus de computadoras causen problemas en redes tales como el internet (Barabási & Bonabeau, Scale-Free Networks, 2003).

A continuación se muestran las métricas que determinan si una red social tiene comportamiento de small world o scale free.

3.3.2 Distribución de network indexes

Average Path Length (Longitud del camino promedio) y Cluster Coefficients (Coeficientes de agrupamiento)

El índice “average path length” (L) se refiere al número de saltos o aristas que unen a cada uno de los nodos o vértices de la red por medio del camino más corto entre ellos. En una red social esto significa que L es el número de amistades promedio en la cadena más corta conectando a dos personas.

De acuerdo a los resultados obtenidos en el estudio (Toriumi, y otros, 2011) de las plataformas de red social analizadas se obtuvo un valor de L promedio de 2.13 con una desviación estándar de 0.339, 56% de los valores de L se encontraban en un intervalo de entre 1.9 y 2.1. Lo que significa que las plataformas de redes sociales tienden a tener valores de L muy pequeños.

El “cluster coefficient” (coeficiente de agrupamiento) o “clustering coefficient” (C) está definido como sigue. Suponiendo que un vértice o nodo v tiene k_v vecinos; entonces a lo mucho $k_v(k_v - 1)/2$ aristas pueden existir entre ellos (ésto ocurre cuando cada vecino de v está conectado a todos los demás vecinos de v). Sea C_v que denota la fracción de esas aristas que son permitidas y que de hecho existen. Define C como el promedio de C_v para todo v. C_v refleja el grado o alcance al cual los amigos de v son también amigos entre sí; y así C mide las asociaciones o la tendencia a asociarse de un típico círculo de amigos. (Strogatz & Watts, 1998).

El valor promedio de C y la desviación estándar son respectivamente 0.377 y 0.206. El valor de C tuvo un amplio rango $0.1 \leq C \leq 0.8$. Sin embargo, aproximadamente el 74% de los valores de C eran mayores que 0.2. Ésto es, las plataformas de redes sociales tienden a tener altos valores de C.

De acuerdo al análisis de estos dos índices se encuentra que la mayoría de las plataformas de redes sociales tienen características de redes small world (Toriumi, y otros, 2011).

Degree Distribution (Distribución de grado)

El "degree" (grado o valencia) de un vértice o nodo en una red es el número de aristas que inciden en ese vértice (Newman, 2003). El esparcimiento del número de aristas que un nodo tiene, está caracterizado por una función de distribución $P(k)$ que da la probabilidad de que un nodo aleatoriamente seleccionado tenga exactamente k aristas (Réka & Barabási, 2002).

Para un gran número de redes, incluyendo la World Wide Web, Internet o redes metabólicas, el grado de distribución tiene una "power-law tail"

$$P(k) \sim k^{-\gamma}.$$

Tales redes son llamadas redes scale free (Réka & Barabási, 2002).

Es posible conocer si una plataforma de red social tiene características de libre escala por medio del análisis de sus distribuciones de grado (Toriumi, y otros, 2011). Sistemas como redes genéticas o la World Wide Web son descritos como redes con una topología compleja. Una propiedad común de las grandes redes es que las conectividades entre vértices o nodos siguen una distribución de "power law" o ley potencial scale free. Esta característica fue encontrada como una consecuencia de dos mecanismos genéricos: redes que se expanden continuamente por la adición de nuevos vértices, y nuevos vértices adjuntos preferentemente en sitios que ya están bien conectados (Barabási & Réka, Emergence of Scaling in Random Networks, 1999).

Para determinar si la distribución de grado sigue una ley potencial, se calculan sus "determination coefficients" (coeficientes de determinación), de la siguiente manera:

$$R^2 = 1 - \frac{\sum_i (\log(y_i) - \log(f_i))^2}{\sum_i (\log(y_i) - \overline{\log(y)})^2},$$

donde $\log(y_i)$ son los valores observados logarítmicamente transformados, $\log(f_i)$ son los valores estimados logarítmicamente de la ley potencial obtenida por el análisis de regresión de la distribución de grado, y $\overline{\log(y)}$ es el promedio de los valores observados transformados logarítmicamente. La ecuación muestra que mientras los valores de R^2 sean más cercanos a 1, la distribución potencial está más cerca de seguir una ley potencial.

El coeficiente de determinación promedio fue de 0.631, y la desviación estándar fue 0.176. Lo

anterior indica que las distribuciones de grado de las plataformas de redes sociales analizadas en el artículo (Toriumi, y otros, 2011) tienden de manera aproximada a seguir una ley potencial. De acuerdo al estudio llevado a cabo en el artículo citado anteriormente de las 615 plataformas analizadas 409 tienen un R^2 mayor a 0.6. El "power index" (índice potencial γ) promedio fue de -0.908, y la desviación estándar de 0.155.

Assortativity

"Assortativity" r, está definido como un índice que muestra el grado de correlación entre nodos conectados. Su rango de valor es $-1 \leq r \leq 1$. Cuando el valor r es mayor, dos nodos que tienen altos grado o "degree" tienden a estar conectados. Por otro lado cuando el valor de r es más pequeño un nodo con alto degree y un nodo con bajo degree tienden a estar conectados.

El valor de r de las plataformas analizadas según (Toriumi, y otros, 2011) es de -0.471 en promedio con una desviación estándar de 0.207, en comparación con plataformas más grandes como mixi que tienen valores para r de 0.121 o Cyworld con -0.13. Los valores de r encontrados muestran que hay una tendencia de que usuarios con altos degrees estén conectados con usuarios con bajos degrees.

Los resultados encontrados por el estudio realizado en el artículo (Toriumi, y otros, 2011) se encuentran en la Tabla 3.1.

Tabla 3.1 Resumen de los valores calculados para los network indexes y sus respectivas desviaciones estándar.

	L	C	R^2	γ	r
Promedio	2.13	0.377	0.631	-0.908	-0.471
Desviación Estándar	0.339	0.206	0.176	0.155	0.207

Fuente: (Toriumi, y otros, 2011).

Estos índices de red muestran que las 615 plataformas de red social analizadas en el artículo (Toriumi, y otros, 2011) tienen características de redes small world, scale free, y assortativity negativa.

3.3.3 Clasificación de plataformas de red social basada en el enfoque de agrupación o clustering approach

De acuerdo al artículo (Toriumi, y otros, 2011) en base a los índices calculados L, C, R^2, y r (γ

no es incluido debido a que muchas plataformas no cumplen con una “power distribution”). Se formula un vector característico v_i,

$$v_i = \left[\frac{L_i}{\sigma_L}, \frac{C_i}{\sigma_C}, \frac{R_i^2}{\sigma_{R^2}}, \frac{r_i}{\sigma_r}\right]$$

Donde $\sigma_L, \sigma_C, \sigma_{R^2}, \sigma_r$ muestran variaciones estándar de promedios de average path length, clustering coefficient, determination coefficients de power law, assortativity, respectivamente. Se aplica el Análisis de Componentes Principales (o PCA por sus siglas en inglés) a la información y entonces se agrupan las plataformas analizadas en cuatro tipos en base a los componentes primarios y secundarios debido a que sus proporciones o razones de contribución son altas. Las razones de contribución de los componentes principales se muestran en la Figura 1 del artículo (Toriumi, y otros, 2011) y un mapeo en dos dimensiones del vector característico para cada plataforma analizada. Posteriormente se aplica el método k-means con un valor de *k* igual a 4. Se nombran a estos cuatro “clusters” o agrupaciones de plataformas como C1 hasta C4.

La Tabla 3.2 muestra información sobre las clasificaciones de los 4 diferentes tipos de redes sociales.

Tabla 3.2 Valores promedio de los network indexes para los cuatro tipos de plataformas de red social.

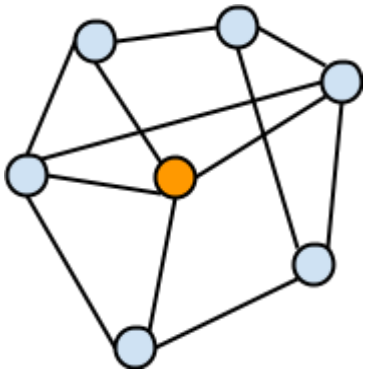
	No. de usuarios	No. de enlaces	*L*	*C*	R^2	*r*	No. de plataformas
C1	283.6	1,001.1	2.095	0.436	0.713	-0.388	263
C2	236.9	287.5	2.025	0.163	0.573	-0.721	184
C3	87.9	743.0	1.833	0.686	0.380	-0.369	92
C4	423.7	1,454.1	2.851	0.313	0.783	-0.280	76

Fuente: (Toriumi, y otros, 2011).

3.3.4 Características de clusters

1. **Plataforma de red social tipo general (C1)**

Figura 3.1 Red social tipo C1.

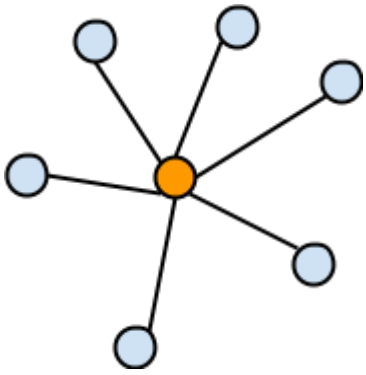


Fuente: (Toriumi, y otros, 2011).

a. 40% de las plataformas de redes sociales analizadas fueron clasificadas como C1.
b. Cuentan con características small world.
c. Presentan características scale free.
d. Tienen valores negativos de assortativity.

2. **Plataforma de red social tipo estrella (C2)**

Figura 3.2 Red Social tipo C2



Fuente: (Toriumi, y otros, 2011).

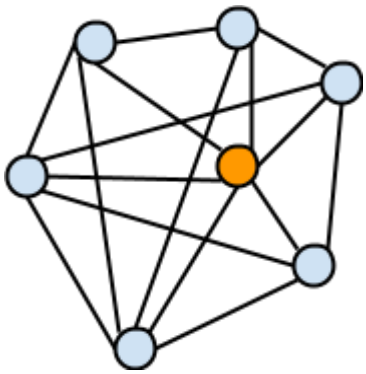
a. Presentan clustering coefficients pequeños.
b. Valores de assortativity pequeños.
c. Degree promedio de 1.21, es decir que muchos de los nodos están conectados a un

solo nodo.

d. El nodo con más conexiones tenía en promedio el 78% de los enlaces en la red, lo que significa que un lado de cada par de nodos estaba casi siempre conectado al mismo usuario.

3. Plataforma de red social tipo agregación (C3)

Figura 3.3 Red social tipo C3.

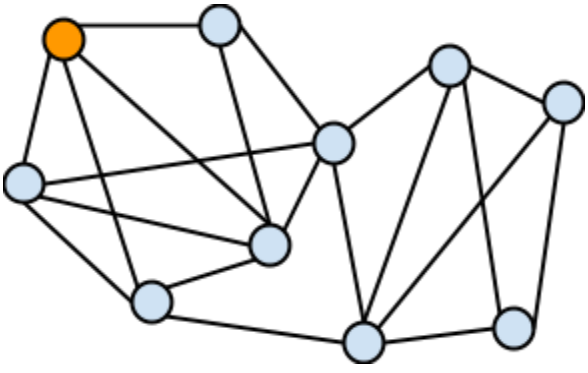


Fuente: (Toriumi, y otros, 2011).

a. Clustering coefficient muy alto.
b. Promedio de longitud de camino más corto bajo.
c. No presenta características de red scale free.
d. Son redes cercanas a un grafo completo en el cual los miembros en esa plataforma están en relaciones intensas (muchos nodos conectados a muchos otros nodos).

4. Plataforma de red social esparcida (C4)

Figura 3.4 Red social tipo C4.



Fuente: (Toriumi, y otros, 2011).

a. Longitud de camino entre nodos larga.

b. Muchos usuarios y usuarios con muchas conexiones de amistad.

3.3.5 Características de Plataformas de red social en base al comportamiento de usuarios

De acuerdo al artículo (Toriumi, y otros, 2011) para la realización de este análisis se deben tener en cuenta 2 condiciones.

1. El número de usuarios dentro de la plataforma de red social debe ser de al menos 100 debido a que el análisis no tendría sentido si el número de usuarios fuese pequeño.
2. Debe haber no solo redes de amigos sino también entradas de blogs o comentarios porque son necesarios para el análisis de la red del comportamiento de la comunicación entre usuarios.

Lo cual implica que el número de plataformas que eran 615 al inicio será recortado a 309 que cumplen con las condiciones anteriores.

Para que sea posible encontrar el comportamiento o los patrones de comportamiento es necesario establecer ciertos índices.

Formulación de índices

1. "Aggregation ratio" (relación o razón de agregación) para amigos A es la razón o proporción de comentarios a amigos en todos los comentarios. Mientras la razón sea mayor, los comentarios para las entradas de blog están más restringidos a los amigos.

$$Aggregation\ ratio\ (A) = \frac{no.de\ comentarios\ por\ amigos}{no.de\ todos\ los\ comentarios}$$

2. "Coverage ratio" (proporción de cobertura) para amigos es la proporción o razón de amigos que publican comentarios en todas las entradas de blog que publican sus amigos. Mientras mayor sea la razón, mayor será el número de comunicaciones que toman lugar en relaciones de amistad.

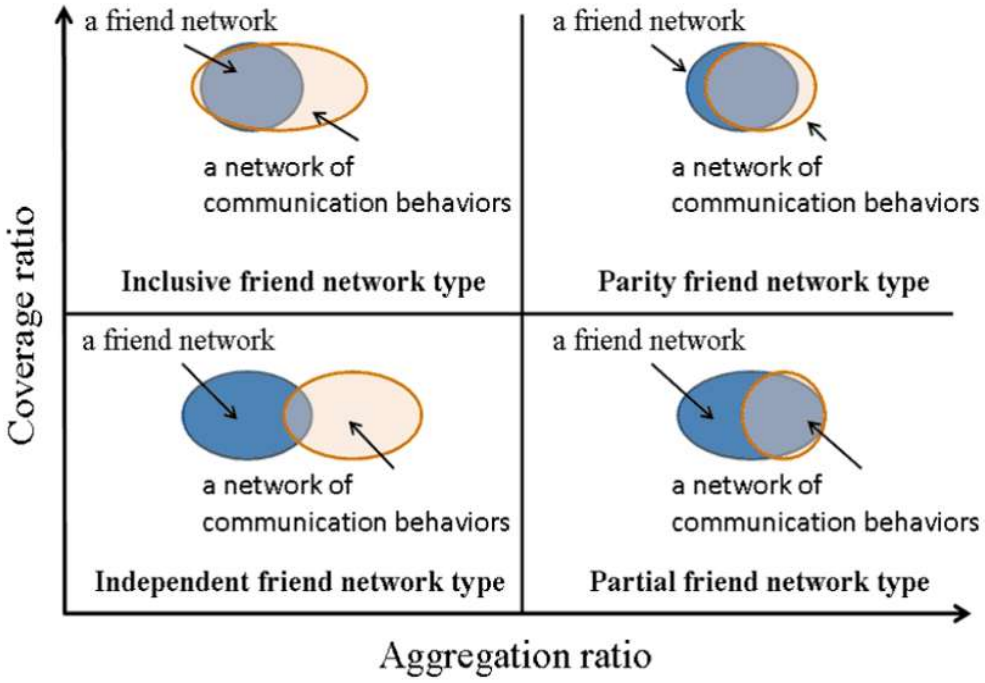
$$Coverage\ ratio\ (C) = \frac{no.de\ amigos\ que\ hacen\ comentarios}{no.de\ amigos\ del\ usuario\ que\ publicaron\ entradas\ en\ el\ blog}$$

Los patrones de comunicación de las plataformas de red social se clasifican sobre la media de los valores de estos dos índices cuyos valores son 0.737 para A y 0.610 para C.

- **Tipo de red "partial friend"**
  Son plataformas sociales con un alto aggregation ratio y un bajo coverage ratio. Los miembros se comunican con solo un grupo limitado de amigos.
- **Tipo de red "parity friend"**
  Son plataformas sociales con altos valores de aggregation ratio y coverage ratio. Los miembros se comunican dentro de sus redes de amigos de manera amplia y exhaustiva, pero muy pocos se comunican con personas fuera de su red de amigos.
- **Tipo de red "inclusive friend"**
  Plataformas sociales con un alto valor de coverage ratio pero un valor bajo de aggregation ratio. Los miembros de esta red se comunican de forma intensa con sus amigos dentro de su red y muchos se comunican con personas fuera de su red de amigos.
- **Tipo de red "independent friend"**
  Plataformas sociales con valores bajos tanto de coverage ratio como de aggregation ratio. Los miembros de estas redes se comunican de manera independiente a su red de amigos.

La Figura 3.5 muestra de manera gráfica lo dicho anteriormente.

Figura 3.5 Cuatro tipos de patrones de comunicación basados en aggregation ratios y coverage ratios.



Fuente: (Toriumi, y otros, 2011)

De acuerdo a lo anterior, las redes sociales pueden ser clasificadas:

1. En base a la estructura de la red.

2. En base al comportamiento de los usuarios.

Para el caso 1 se definieron 4 clasificaciones: C1, C2, C3 y C4. Estas clasificaciones indican las características que cada estructura de red presenta, si tiene características small world o scale free, si contienen un gran número de enlaces entre nodos, entre otras. Para el caso 2 se definen 4 clasificaciones que indican cómo es la interacción social entre los usuarios: inclusive friend, parity friend, independent friend, partial friend, estas clasificaciones indican si los usuarios tienen grupos limitados de amigos, si se comunican con personas externas a sus grupos de amigos, además de otras.

Definir en qué clasificación ya sea en base a la estructura de la red o en base al comportamiento de los usuarios se pueda encontrar el concepto de plataforma de red social planteado en este proyecto es algo difícil de definir, debido a que, esa clasificación solo puede ser hecha por medio del análisis de los network indexes, por tanto, no es posible por el momento clasificarla dentro de alguna de las clasificaciones anteriores dado el hecho de que la aplicación no ha sido liberada al público y por lo tanto no cuenta con usuarios reales.

Los Capítulos 1, 2 y 3 fueron capítulos introductorios, los siguientes capítulos muestran lo concerniente al desarrollo de la aplicación como el modelado de los datos, los patrones de diseño empleados para su realización, las tecnologías usadas para el desarrollo de la misma, entre otros temas más.

Capítulo 4 Tecnologías para el desarrollo de aplicaciones empresariales

En este capítulo se describirán las tecnologías utilizadas para la implementación y modelado del REST API.

4.1 Manejadores de Bases de Datos

Se utilizaron dos tipos de manejadores de Base de Datos que por sus características las hacen adecuadas para resolver dos problemas distintos de almacenamiento de información. HSQLDB es un manejador de base de datos relacional desarrollado en Java que puede ser embebido junto con la aplicación y reside en memoria. MongoDB es un manejador de base de datos no relacional NoSQL orientado a documentos que es particularmente útil para almacenar datos que no están fuertemente relacionados entre sí y que proporciona un excelente desempeño en el almacenamiento y recuperación de datos binarios.

4.1.1 HSQLDB

HSQLDB es un manejador de base de datos relacional de alto rendimiento escrito en el lenguaje de programación Java que soporta los modos: embebido (es decir, que está estrechamente integrado con un software para desarrollo de aplicaciones que requieren acceso a información almacenada) y servidor, además de almacenamiento de tablas en memoria o en disco. Se distribuye bajo la licencia del Hypersonic SQL Group la cual está basada en la licencia BSD, por lo tanto, es software completamente libre. Soporta el estándar ANSI-92 SQL casi en su totalidad. Es mantenido por The hsql Development Group.

HSQLDB ha sido desarrollado de manera constante por más de 12 años y es usado como una base de datos y motor de persistencia en más de 1,700 proyectos de software open source y muchos productos comerciales.

La ventaja de este tipo de manejador de base de datos embebible es su versatilidad, permitió desarrollar el prototipo de la aplicación de forma rápida, además se puede distribuir junto con la aplicación evitando depender de servidores y configuraciones externas (The hsql Development Group, 2014).

4.1.2 MongoDB

MongoDB es un manejador de base de datos de código abierto orientado a documentos o NoSQL

escrito en C++, entre sus características principales están las siguientes:

- Almacenamiento orientado a documentos mediante documentos estilo JSON.
- Soporte para índices sobre cada atributo.
- Auto-Sharding. Sharding es el proceso del almacenamiento de registros de datos a través de múltiple máquinas.
- Consultas basadas en documentos.
- MapReduce. Manejo de grandes cantidades de datos.
- GridFS. Es una especificación para el almacenamiento y recuperación de archivos que excedan el tamaño límite de 16 MB para documentos BSON (un formato de serialización utilizado para almacenar documentos en MongoDB). En lugar de almacenar un archivo en un solo documento, GridFS divide un archivo en partes y almacena cada una de estas partes como un documento separado. De esta manera es posible accesar información de manera arbitraria de cualquiera de las secciones del archivo. GridFS es útil también para almacenar archivos a los cuales se desea accesar sin necesidad de cargar el archivo entero en memoria.
- Alto desempeño.

MongoDB es ideal para ser utilizado como repositorio de documentos debido a las características listadas anteriormente (MongoDB, Inc., 2013).

4.2 Definición de ORM

Object Relational Mapping (ORM) o Mapeo Objeto Relacional es una técnica de programación que tiene como finalidad el mapeo en ambas direcciones de un modelo de datos Java o cualquier otro lenguaje de programación orientado a objetos al correspondiente modelo relacional de base de datos. De manera que, el programador no requiere conocer los detalles sobre la base de datos, lo cual da grandes ventajas, pero a su vez, el uso de un ORM genera ciertas desventajas.

Ventajas

- Independencia del manejador de base de datos. Reemplazar un manejador de base de datos relacional por otro manejador se vuelve una tarea muy sencilla, pudiendo así probar distintos manejadores y eligiendo el que mejor se adecúe a las necesidades del proyecto.
- Facilidad de acceso a los datos. Otra de las grandes ventajas que brinda el mapeo ORM es la facilidad de acceso a los datos existentes en la base de datos, por medio del código que contiene la lógica de negocio, manteniendo así, la ventaja de la programación orientada a objetos, puesto que, no se accede la base de datos directamente, sino que, se acceden los objetos que se recuperan de ella por medio del mapeo ORM.

Desventajas

- El mapeo ORM conlleva algunas dificultades, las cuales son conocidas con el término en inglés “Object Relational Impedance Mismatch” (Diferencia de Impedancia Objeto Relacional), algunas de estas dificultades se refieren principalmente a los conceptos del paradigma orientado a objetos que no existen en el modelo relacional, y a las diferencias existentes entre los tipos de datos que se manejan del lado del lenguaje de programación y del lado del manejador de la base de datos, entre otras. Más adelante se habla con mayor detalle sobre este concepto.
- Al diseñar un modelo de datos en el POO que será traducido al modelo relacional por medio de un framework, se pierde el control sobre la implementación de ese modelo, algunas veces el modelo de datos no será eficiente desde el punto de vista del modelo relacional.

4.2.1 Diferencia de Impedancia Objeto Relacional

El Object Relational Impedance Mismatch se refiere a los conceptos lógicos que existen en un modelo de datos en este caso el modelo relacional, pero que, cuya equivalencia no existe en el Paradigma Orientado a Objetos (específicamente para el caso del lenguaje de programación Java) y viceversa (Red Hat, 2014).

1. **Granularidad**. Algunas veces el modelo orientado a objetos tiene más clases que el número de tablas correspondientes en la base de datos relacional (El modelo de objetos es más granular que el modelo relacional).
2. **Subtipos** (Herencia). La herencia es un concepto natural en la programación orientada a objetos. Sin embargo, el modelo relacional no define nada como tal.
3. **Identidad**. El modelo relacional define un único concepto de identidad que son las llaves primarias. Java, sin embargo, define dos conceptos: identidad de objeto por medio del operador “==” (a == b) e igualdad de objetos por medio del método “equals()” (a.equals(b)).
4. **Asociaciones**. En los lenguajes orientados a objetos las asociaciones son representadas como referencias unidireccionales, en contraste con el modelo relacional que hace uso de llaves foráneas. Si es necesaria una relación bidireccional en Java, se debe definir la asociación dos veces.
5. **Navegación de datos**. La manera de accesar a los datos en Java es completamente diferente a la manera en la que se acceden los datos en una base de datos relacional. En Java los datos se navegan de una asociación a otra recorriendo toda la red de objetos asociados.

   Lo anterior no es una forma eficaz de recuperar datos de una base de datos relacional. Por lo general, se desea reducir al mínimo el número de consultas SQL y, por tanto cargar

varias entidades a través de operadores JOIN y seleccionar las entidades antes de empezar a caminar por la red de objetos.

4.3 Java Persistence API

El Java Persistence API (JPA por sus siglas en inglés) o API de Persistencia de Java, provee un modelo de persistencia basado en POJOs para mapeo objeto relacional. JPA fue desarrollado por el grupo de expertos en software EJB 3.0 como parte del JSR 220. Puede ser utilizado en la plataforma Java EE y en la plataforma Java SE.

JPA está compuesto por una serie de aspectos que lo convierten en una herramienta muy poderosa y flexible, algunas de sus características principales de acuerdo con (Keith & Schincariol, 2009) son:

- **Persistencia POJO**. Es quizás el aspecto más importante de JPA, los objetos que son hechos persistentes no tienen ninguna característica especial. Cualquier objeto que tenga un constructor por default puede ser un objeto persistente. El mapeo objeto relacional se realiza por medio de metadatos, utilizando XML en un archivo externo o anotaciones Java dentro del código.
- **No intrusivo**. El API de persistencia existe como una capa separada de los objetos persistentes. El API de persistencia es llamado por la lógica de negocio de la aplicación y le son pasados los objetos de persistencia y la manera en la que debe de operar sobre ellos.
- **Consultas de objeto**. JPA ofrece la posibilidad de realizar consultas a las entidades y asociaciones o relaciones entre entidades modeladas. Las consultas pueden ser expresadas por medio del lenguaje JPQL, este lenguaje no está atado al esquema de la base de datos, lo que significa que las entidades Java y sus atributos son usados como el esquema para consultas y no la base de datos directamente, por lo tanto, no se requiere conocer información sobre la relación entre la base de datos y el mapeo de la base de datos. Eventualmente las consultas desarrolladas con JPQL serán traducidas por el framework que implementa el JPA a consultas SQL que serán ejecutadas directamente en la base de datos relacional.
- **Entidades móviles**. Los objetos deben ser capaces de ser movidos de una máquina virtual a otra y poder regresar a la máquina virtual de la que salieron y aún así ser útiles dentro de la aplicación. A manera de satisfacer los requerimientos necesarios para las aplicaciones distribuidas.
- **Simple configuración**. Como se mencionó anteriormente la configuración de la persistencia del JPA puede ser hecha de dos maneras, en realidad tres, por medio de un archivo XML para separar los metadatos del código, por medio de anotaciones Java que se incluyen dentro del código, y la tercera manera que involucra el uso de anotaciones

Java y el uso de XML al mismo tiempo. JPA hace uso de valores por default, lo que implica que la configuración requerida es mínima y si los valores por default propuestos por JPA no son suficientes para los requerimientos que se exijan, aún así la cantidad de metadatos que se utilicen para la configuración de dichas exigencias será mínima.

4.4 Hibernate

Hibernate es un framework que implementa el JPA y realiza el mapeo objeto relacional (ORM), es decir, mapea la información del modelo orientado a objetos al modelo relacional (objetos a registro) y viceversa (registro a objeto). Hibernate permite el mapeo de conceptos de la POO al modelo relacional. Hibernate, además, es software libre y se distribuye bajo la licencia LGPL.

Hibernate cuenta con una serie de ventajas, entre las cuales están las siguientes y que se encuentran en la página oficial del framework (Red Hat, 2014):

- Hibernate permite al desarrollador detallar el modelo de datos, las relaciones que existen entre las diversas entidades que conforman el modelo de datos y su estructura, con base en esta información Hibernate permite a la aplicación manipular los registros de la base de datos en forma de objetos, con todos los conceptos que involucra la POO.
- Hibernate convertirá los tipos de datos utilizados por Java a tipos de datos utilizados por el manejador de la base de datos y viceversa.
- Genera las consultas SQL, liberando de esa manera al desarrollador del manejo manual de los datos que generó la consulta ejecutada, logrando la portabilidad entre todos los manejadores de bases de datos.
- Hibernate genera el SQL apropiado en base al dialecto que se esté usando en la configuración, existe una gran cantidad de dialectos que Hibernate soporta. Existen dialectos para al menos los manejadores de bases de datos principales.
- Aparte de poder crear un modelo de datos relacional tomando como base un modelo de datos orientado a objetos, Hibernate también brinda la posibilidad de crear un modelo de datos orientado a objetos teniendo ya desarrollado un modelo de datos relacional.
- Cuenta con un lenguaje de consultas llamado HQL y un API para el desarrollo de consultas de manera programática llamado Criteria API.
- Hibernate puede ser utilizado en la plataforma Java SE y la plataforma web Java EE.
- Es un framework de alto desempeño.

4.5 Servicios Web REST

Un servicio web es un método de comunicación que utiliza un conjunto de protocolos para lograr dicha comunicación. Generalmente el protocolo que más se utiliza para desempeñar el

intercambio de información por medio de servicios web en las aplicaciones es el protocolo HTTP, aunque dicho protocolo no es el único sobre el cual puede realizarse la comunicación.

La principal ventaja de los servicios web es que brindan independencia de la plataforma, es decir, que podría lograrse la integración entre una aplicación web desarrollada en Java que se está ejecutando sobre un servidor web con una aplicación que ha sido desarrollada en Objective-C que se ejecuta sobre un cliente que es un dispositivo móvil, este es un ejemplo del uso de REST API para lo que se conoce como Integración de Aplicaciones.

4.5.1 REST (Representational State Transfer)

El contenido de esta sección ha sido recopilado de la disertación de Roy Thomas Fielding (Fielding, 2000).

REST es una arquitectura para sistemas hipermedia. Se deriva de los sistemas basados en red como el modelo cliente servidor. Para la arquitectura REST las necesidades del sistema son vistas como un todo, teniendo como enfoque el entendimiento del contexto del sistema. Las siguientes son las restricciones de la arquitectura REST.

1. **Cliente-Servidor**. La separación de contenidos es el principio detrás del modelo cliente-servidor. Separando la interfaz de usuario del almacenamiento de la información se mejora la interfaz de usuario a través de las múltiples plataformas y además se mejora la escalabilidad por medio de la simplificación de los componentes del lado del servidor. La separación permite a los componentes seguir evolucionando de manera independiente.
2. **Stateless.** Cada solicitud del cliente al servidor debe contener toda la información necesaria para entender la solicitud, y no puede tomar ventaja de ningún contexto almacenado en el servidor. El estado de la sesión debe ser guardado en el lado del cliente. Esta restricción trae consigo las siguientes propiedades:

    a. Visibilidad. El monitoreo del sistema no tiene que ver más allá de una parte de una sola solicitud con el fin de determinar la verdadera naturaleza de la solicitud.
    b. Confiabilidad. Facilita la tarea de recuperarse de fallas parciales.
    c. Escalabilidad. El no tener que almacenar información de sesiones permite a los componentes del servidor liberar recursos de manera más rápida. Además, facilita la implementación porque el servidor no tiene que hacer uso del manejo de recursos en las distintas solicitudes.

    La desventaja que acarrea es que puede tener impacto en el desempeño de la red debido al incremento de información repetida. En contraste a ésto, colocar la información de la sesión del lado del cliente hace que la aplicación se vuelva dependiente de la correcta

implementación de la semántica en los diferentes clientes.

3. **Caché**. La información dentro de la respuesta a la solicitud debe estar etiquetada como cacheable o no cacheable. Si una respuesta es cacheable, entonces se le da el derecho a un "cache client" para volver a usar la información de la respuesta más adelante cuando existan solicitudes equivalentes. De ésta manera se mejora la eficiencia de la red puesto que se pueden eliminar algunas interacciones dentro del sistema, mejorando también, escalabilidad, y el desempeño que el usuario percibe. La desventaja es que el caché puede disminuir la confiabilidad si información viciada difiere significativamente de la información que se habría obtenido si hubiera sido enviada la petición directamente al servidor.
4. **Interfaz uniforme**. La característica central de REST es la interfaz uniforme entre componentes. La implementación está desacoplada de los servicios que provee, lo que lleva a que exista la capacidad de evolución independiente de los componentes. La desventaja es que la interfaz uniforme degrada la eficiencia, ya que la información es transmitida en una forma estandarizada y no de acuerdo a las necesidades específicas de la aplicación. La interfaz REST está diseñada para ser eficiente para transferencia de datos hipermedia, optimizando para la web, pero resultando en una interfaz que no es óptima para otras formas de arquitectura. REST está definido por cuatro conceptos: Identificación de recursos; manipulación de recursos por medio de representaciones; mensajes autodescriptivos; e, hipermedia como el motor de estado de la aplicación.
5. **Sistema por capas**. Mejora el comportamiento para los requerimientos de internet a gran escala, el estilo de sistema por capas permite a una arquitectura estar compuesta de capas jerárquicas restringiendo el comportamiento de los componentes de tal manera que cada componente no puede "ver" más allá de la capa inmediata con la cual ellos están interactuando. Restringiendo el conocimiento del sistema a una sola capa, se establece un límite sobre la complejidad del sistema y se promueve la independencia de cada capa. La desventaja principal de los sistemas por capa es que agregan sobrecarga y latencia al procesamiento de la información, reduciendo el desempeño percibido por el usuario.
6. **Código en demanda**. Permite que la funcionalidad del lado del cliente se extienda descargando y ejecutando código en la forma de applets o scripts. Ésto simplifica los clientes al reducir el número de características requeridas para ser preimplementadas. Permitir que las características sean descargadas después del desplegado mejora la extensibilidad del sistema. Sin embargo, también reduce la visibilidad, y debido a ésto es una restricción opcional de REST.

4.5.2 El enfoque RESTful

El enfoque RESTful se refiere a los principios de diseño para el desarrollo de servicios web basados en REST. En base a las condiciones vistas en la sección anterior se pueden resumir las características que debe de tener un API de servicios web para ser considerado RESTful.

- Usar métodos HTTP.
- Comunicación stateless.
- Orientado a recursos. La abstracción clave de la información en REST es un recurso.
- Darle a todos los recursos un "id" por medio de URIs.
- Debe proveer múltiples representaciones de recursos para múltiples necesidades.
- Hipermedia como el motor de estado de la aplicación.
- Transferir XML, JavaScript Object Notation (JSON), o ambos.
- Los recursos se nombran con un sustantivo, los verbos se refieren a métodos (GET, POST, DELETE, etc).

Los servicios web basados en REST han cobrado auge en los últimos años debido a su simpleza en comparación con las interfaces diseñadas que utilizan SOAP. Muchas de las plataformas sociales más grandes del mundo como Twitter o Facebook hacen uso de REST APIs, también plataformas para el comercio electrónico como Amazon. Para ejemplificar el uso de los conceptos sobre REST se listan a continuación algunos ejemplos que hacen uso de REST APIs consolidados.

Twitter REST API version 1.1

Regresa una colección de los Tweets más recientes por el usuario indicado por el parámetro screen_name.

https://api.twitter.com/1.1/statuses/user_timeline.json

Ejemplo:

https://api.twitter.com/1.1/statuses/user_timeline.json?screen_name=twitterapi&count=2

Véase (Twitter, 2013).

Facebook Graph API version 2.0

Servicio de lectura que regresa un usuario identificado por el parámetro /{user-id} que representa a una persona en Facebook.

https://graph.facebook.com/{user-id}

Ejemplo:

https://graph.facebook.com/me

Véase (Facebook, 2014).

4.5.3 JSON y Jackson

JSON (JavaScript Object Notation) es un formato de intercambio de datos muy popular como formato de transferencia para servicios web. Está basado en un subconjunto del estándar ECMA-262 Tercera Edición del lenguaje de programación JavaScript. JSON es un formato de texto que es completamente independiente del lenguaje (Introducing JSON, s.f.).

JSON está construido sobre dos estructuras:

- Una colección de pares nombre/valor.
- Una lista ordenada de valores.

La aplicación que ha sido desarrollada como parte de este proyecto utiliza JSON como formato de entrada y salida, debido a que Java no soporta JSON de manera nativa, requiere de librerías para la traducción de objetos JSON a objetos Java y viceversa. Existen numerosas librerías para realizar dicha traducción, sin embargo, para el desarrollo de este proyecto se optó por la librería Jackson.

Jackson es una librería Open Source de alto rendimiento para el procesamiento de datos en formato JSON, Nos permite hacer conversiones de objetos entre JSON y Java, ha sido distribuido bajo las licencias Apache License 2.0 y LGPL 2.1

4.6 El Framework Spring

Spring es un framework para el desarrollo de aplicaciones empresariales que corren en plataforma Java. Soporta varios lenguajes de programación que corren sobre la JVM (Java Virtual Machine), como Groovy, JRuby, Jython y por supuesto Java. Spring es un Full Stack Framework, lo cual significa que provee todas las herramientas que una aplicación web empresarial requiere, incluye una serie de características que lo convierten en un framework de desarrollo de aplicaciones muy poderoso, algunas de estas características son las siguientes y se encuentran citadas en la página de internet oficial del mismo framework (GoPivotal, Inc., 2014):

- Inyección de dependencias con dos estilos de configuración: XML y basada en anotaciones Java o código Java.
- Soporte avanzado para programación orientada a aspectos basada en proxy y variantes basadas en AspectJ incluyendo soporte para manejo declarativo de transacciones.
- Provee poderosas abstracciones para trabajar con especificaciones comunes de Java EE tales como JDBC, JPA, JTA y JMS.
- Soporte de primera clase para frameworks open source como Hibernate y Quartz.
- Un web framework flexible para la construcción de aplicaciones RESTful MVC y service

endpoints.
- Facilidades para testing por pruebas unitarias así como también para pruebas de integración.
- Un framework de seguridad que soporta estándares para seguridad de aplicaciones como Basic Authentication, OAuth, etc.
- Mecanismos y soporte de “caching” para mejorar el desempeño de las aplicaciones.

Spring Framework es modular en su diseño, por tanto, permite la adición de nuevos módulos o proyectos al framework. Existen una gran cantidad de proyectos que conforman el Spring Framework como lo muestra la página oficial del mismo, algunos de ellos son:

- Spring Security
- Spring Mobile
- Spring Data
- Spring Web Services
- Spring Batch
- Spring Integration
- Spring Social
- Spring Web Flow
- Spring ROO
- Spring MVC
- Spring Boot

Los proyectos que conforman parte del Spring Framework pueden ser consumidos de manera individual. Spring Framework soporta un amplio rango de plataformas para el desplegado de aplicaciones independientes para Tomcat y para servidores Java EE. Spring puede ser desplegado sobre las plataformas cloud principales con soporte para Java como Heroku, Google App Engine, Amazon Elastic Beanstalk y Cloud Foundry.

Spring Framework se distribuye bajo la licencia Apache versión 2.0.

4.6.1 Dependency Injection

Spring hace uso del concepto “Dependency Injection” (DI) o Inyección de dependencias. Algunas veces este término es intercambiable con el principio de “Inversion of Control” (IoC) o Inversión de Control. Cuando un programa se ejecuta, primero corre el programa principal. Cuando se usa el principio IoC, las dependencias entre objetos son manejadas y resueltas por un contenedor, el cual se encarga de crear las instancias de los objetos e inyectar sus dependencias con otros objetos, a esto último se le conoce como "Object Wiring" (alambrado de objetos). El programa entonces está listo para continuar con su ciclo de funcionamiento. Se puede apreciar el

porqué del nombre Inversion de Control, ya que comúnmente el objeto mismo es el que tiene el control de cómo crear sus propias dependencias (Konda).

En Spring, los objetos que forman la columna vertebral de las aplicaciones y que son manejados por el contenedor Spring IoC son llamados “beans”. Un “bean” es un objeto que es instanciado, ensamblado y manejado por el contenedor Spring IoC.

Un contenedor IoC se puede configurar usando una o varias tecnicas: archivos XML, anotaciones Java, clases de configuración. Dicha configuración define los beans que se habrán de instanciar y las dependencias entre ellos (Johnson, Hoeller, Donald, & Sampaleanu, 2014).

Además el uso de IoC facilita el testing automático de código, ya que se pueden inyectar objetos Mock para usarse en tests unitarios, más adelante en la sección 4.7 se habla a detalle sobre esta característica (Johnson, Hoeller, Donald, & Sampaleanu, 2014).

Es precisamente el problema del “coupling” entre clases el principal problema que resuelve Spring por medio de la inyección de dependencia. El término coupling se refiere al grado al cual una clase conoce sobre otra clase.

La Figura 4.1 es una vista de manera general sobre el funcionamiento de Spring. Las clases de la aplicación son combinadas con la configuración (metadatos) de tal manera que después de que el contenedor IoC ha sido creado e inicializado, se tiene un sistema o aplicación totalmente configurada y ejecutable.

Figura 4.1 El contenedor Spring IoC

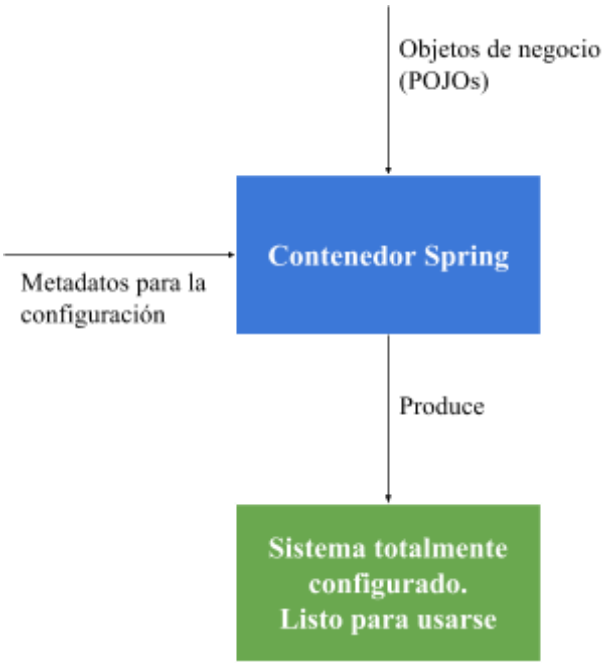


Figura 4.1. El contenedor Spring IoC.

4.6.2 Spring MVC

El framework Spring Web Model-View-Controller (MVC) está diseñado en torno a un DispatcherServlet que despacha solicitudes a manejadores, con mecanismos de mapeo configurables, resolución de vistas, locale, zonas horarias, etc. también tiene soporte para la carga o subida de archivos. El manejador por default está basado en las anotaciones @Controller y @RequestMapping, ofreciendo un amplio rango de métodos de manipulación flexibles. Desde la introducción de Spring 3.0, el mecanismo @Controller también permite crear sitios Web y aplicaciones RESTful, a través de la anotación @PathVariable.

Algunas de las características que soporta el framework son las siguientes:

- Separación de roles. Cada rol (controlador, validador, DispatcherServlet, entre otros.) puede ser realizado por un objeto especializado.
- Configuración sencilla y poderosa tanto del mismo framework como de las clases de la aplicación.
- Adaptable, no intrusivo y flexible. Permite la definición de cualquier método en los controladores por medio de una variedad de anotaciones.
- Código de negocio reusable. Permite el uso de objetos de negocio existentes.
- Validación personalizable.

- Manejadores de mapeo personalizables y resolución de vistas.
- Modelo de transferencia flexible. Transferencia de modelo con un mapa nombre/valor y soporte de integración con cualquier tecnología para la Vista.
- Locale personalizable, zona horaria, soporte para JSPs con tag library, soporte para JSTL (JSP Standard Tag Library), etc.

Véase la página de la referencia oficial de Spring MVC (Pivotal, 2014).

4.6.3 Spring Security

Spring Security es un framework de autenticación y control de acceso muy poderoso y altamente personalizable. Es el estándar para el aseguramiento de aplicaciones basadas en Spring (Alex & Taylor, s.f.).

La seguridad es sin duda uno de los componentes arquitecturales más críticos en el desarrollo de aplicaciones web, en especial porque las aplicaciones web se encuentran expuestas a internet. Por ello Spring Security implementa un conjunto de servicios para la autenticación y control de acceso para aplicaciones. Hay principalmente dos áreas que son el objetivo de Spring Security.

- **Autenticación** que es el proceso de establecer si un principal es quien dice ser (un "principal" es un término en inglés para referirse a un usuario, dispositivo o algún otro sistema que puede desempeñar una acción dentro de la aplicación).
- **Autorización** que se refiere al proceso de decidir si el principal está autorizado para desempeñar una acción dentro de la aplicación. Para llegar al punto donde una decisión de autorización debe ser hecha, la identidad del principal ya debe haber sido establecida por el proceso de autenticación.

Algunas de las características principales que ofrece Spring Security son las siguientes:

- Fácil configuración utilizando inyección de dependencias.
- Configuración no intrusiva.
- No invasivo. Mantiene a los objetos de la aplicación libre de código de seguridad. A menos que se especifique la interacción con el contexto de seguridad.
- Soporte para OpenID.
- Soporte para autenticación HTTP BASIC. Spring Security puede procesar directamente solicitudes de autenticación HTTP BASIC.
- Soporte para autenticación HTTP Digest.
- Fácil integración con bases de datos existentes.
- Codificación de passwords. Soporte para codificación SHA y MD5.
- Soporte para Tag Library.

4.6.4 Spring Data

Spring data es un framework que forma parte del conjunto de proyectos que componen al Spring Framework. Como todos los demás proyectos Spring, Spring Data es un proyecto Open Source. Spring Data hace más sencilla la construcción de aplicaciones desarrolladas utilizando Spring que hacen uso de las nuevas tecnologías de acceso a la información tales como bases de datos no relacionales, frameworks de tipo MapReduce (MapReduce es un modelo de programación y una implementación para el procesamiento y generación de grandes conjuntos de datos), servicios de información basados en tecnologías cloud así como proveer soporte mejorado para tecnologías de bases de datos relacionales. Véase la página oficial del Framework Spring Data (GoPivotal, 2014).

Spring Data soporta una gran variedad de tecnologías, pero la que en realidad fue importante para el desarrollo del proyecto fue el soporte para JPA y MongoDB.

El poder de Spring Data radica en la facilidad que brinda para el rápido desarrollo de consultas al modelo persistente. La idea básica de cómo Spring Data permite un rápido desarrollo de consultas es mediante la obtención de toda la información que es requerida para ejecutar una consulta del nombre del método declarado en una interfaz que se marca con la anotación @Repository esta anotación es un estereotipo de Spring que sirve para decorar objetos que tienen acceso a una base de datos. En tiempo de ejecución Spring inyecta una implementación que crea y ejecuta la consulta vía el JPA Criteria API. Es decir, Spring Data crea y ejecuta automáticamente la consulta partiendo del nombre de un método, puede haber casos en los que la consulta sea tan compleja que no pueda hacerse por medio de la definición de un nombre, para ello existe la anotación @Query, de tal forma que cuando el desarrollador necesite escribir él mismo la consulta puede hacerlo al declarar un método dentro de la interfaz marcada con @Repository y anotarlo con @Query para después definir la consulta como un parámetro de la anotación.

A continuación se realiza una comparación entre una consulta tradicional en JPA por medio de Named Queries y otra consulta utilizando Spring Data JPA.

Consulta tradicional usando JPA:

```
@Entity
@NamedQuery( name="query", query = "SELECT u FROM User u where u.name = :name" )
public class User {
        ...
}
```

```
@Repository
public class UserRepository {

	@PersistenceContext
	EntityManager em;

	public List<User> findByName(String name) {
		TypedQuery<User> q = getEntityManager().createNamedQuery("query", User.class);
		q.setParameter("name", name);

		return q.getResultList();
	}

	...

}
```

Nótese como la entidad User está estrechamente acoplada a la consulta SQL, además se debe definir la lógica para la obtención de los datos recuperados por la ejecución de la consulta.

Spring Data por su parte permite la Separación de Responsabilidad (SoC) por medio de la definición de una interfaz en este caso UserRepository que está marcada con el estereotipo @Repository y que contiene todas las consultas que se realicen sobre la entidad User. Al usar Repositories de Spring Data, Spring está creando un DAO (ver Capítulo 5 para definición de DAO). Claramente se percibe en el ejemplo la simplicidad que Spring Data introduce en una aplicación.

Consulta usando Spring Data JPA:

```
@Repository
public interface UserRepository extends JpaRepository<User, String> {

	// Spring Data infiere el query en base al nombre del método
	List<User> findByName(String name);

}
```

Spring Data infiere la consultas por medio de los nombres en los métodos abstractos definidos en la interfaz marcada con @Repository. La siguiente tabla muestra la sintaxis que sigue dicha inferencia.

Tabla 4.1 Tabla de sintaxis para la inferencia de queries usando Spring Data.

Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname = ?2
Is, Equals	findByFirstname, findByFirstnameIs, findByFirstnameEquals	… where x.firstname = 1?
Between	findByStartDateBetween	… where x.startDate between 1? and ?2
LessThan	findByAgeLessThan	… where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	… where x.age <= ?1
GreaterThan	findByAgeGreaterThan	… where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	… where x.age >= ?1
After	findByStartDateAfter	… where x.startDate > ?1
Before	findByStartDateBefore	… where x.startDate < ?1
IsNull	findByAgeIsNull	… where x.age is null
IsNotNull, NotNull	findByAge(Is)NotNull	… where x.age not null
Like	findByFirstnameLike	… where x.firstname like ?1
NotLike	findByFirstnameNotLike	… where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	… where x.firstname like ?1 (parameter bound with appended %)
EndingWith	findByFirstnameEndingWith	… where x.firstname like ?1 (parameter bound with prepended %)
Containing	findByFirstnameContaining	… where x.firstname like ?1 (parameter bound wrapped in %)
OrderBy	findByAgeOrderByLastnameDesc	… where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	… where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	… where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> age)	… where x.age not in ?1
True	findByActiveTrue()	… where x.active = true

False	findByActiveFalse()	… where x.active = false
IgnoreCase	findByFirstnameIgnoreCase	… where UPPER(x.firstame) = UPPER(?1)

Fuente: (Gierke, Darimont , & Strobl, 2014)

Spring Data también cuenta con soporte para sistemas de bases de datos no relacionales como es el caso de MongoDB. Spring Data permite el acceso a datos dentro de MongoDB por medio de repositorios al igual como lo hace con JPA. También brinda soporte para GridFS.

4.7 Pruebas automatizadas por software

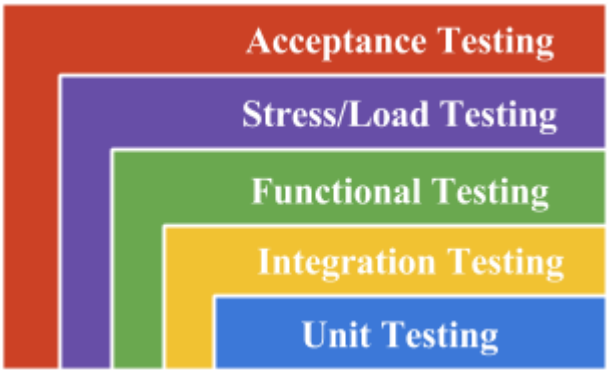
La automatización de pruebas por software es un proceso en el desarrollo de software en el que son ejecutados varios scripts o "tests" por un software especial para controlar la ejecución de los mismos sobre una aplicación, antes de que ésta sea liberada en la etapa de producción y con la finalidad de comparar los resultados arrojados por los tests con resultados predichos o esperados por el desarrollador. Las pruebas automatizadas facilitan tareas repetitivas pero que son necesarias y que serían difíciles de desempeñar manualmente.

Las pruebas de software son un elemento crítico en el aseguramiento de la calidad del software y representan la última revisión de la especificación, diseño y generación de código, además de que el proceso de escritura de los tests o pruebas incrementan drásticamente la habilidad del programador para desarrollar objetos que se comporten de manera adecuada.

1. Las pruebas de software tienen como objetivo la ejecución de un programa con la intención de encontrar un error.
2. Un buen "test case" o caso de prueba es aquél que tiene una alta probabilidad de encontrar un error, aún sin descubrir.
3. Una prueba exitosa es aquella que descubre un error, aún no descubierto.

Existen diferentes tipos de pruebas de software y pueden ser categorizadas de distintas maneras, la Figura 4.2 describe una de las tantas categorizaciones que pueden existir.

Figura 4.2 Los diferentes tipos de tests.



Fuente: (Tahchiev, Leme, Massol, & Gregory)

La Figura 4.2 muestra los 5 tipos diferentes de "software testing" o pruebas de software. Los tests dentro de las cajas exteriores son más amplios en su alcance. Los tests en las cajas interiores son de un alcance más reducido. A medida que se avanza de las cajas interiores a las exteriores, las pruebas se vuelven cada vez más funcionales y requieren que una mayor parte de la aplicación esté presente (Tahchiev, Leme, Massol, & Gregory). Sin embargo, para el desarrollo de este proyecto de tesis se utilizaron el Integration Testing y el Unit Testing que corresponden a las dos cajas más interiores de la Figura 4.2.

A continuación se detalla de manera breve en qué consiste cada una de las pruebas implementadas dentro del proyecto de tesis.

4.7.1 Unit Testing (Pruebas Unitarias)

"Un *unit test* examina el comportamiento de una *unidad de trabajo*. Dentro de una aplicación Java se entiende que el término *unidad de trabajo* hace referencia a menudo a un solo método. Además una *unidad de trabajo* es una tarea que no es directamente dependiente de la realización de ninguna otra tarea." (Tahchiev, Leme, Massol, & Gregory).

De manera genérica, un unit test es un programa que confirma que el método acepte el rango de entrada esperado y que el método retorne el valor esperado para cada entrada. Es decir, que un unit test es una evaluación de como el programador espera que funcionen los métodos que ha desarrollado. Así entonces, nos permite probar el comportamiento de un método a través de su interfaz. Si se le da un valor x, ¿retornará un valor y? Si se le da un valor z en su lugar, ¿tirará la excepción esperada?

El objetivo principal del unit testing es verificar que las aplicaciones funcionen de manera adecuada y como se tiene esperado, además de la captura de bugs o errores dentro de la

aplicación de manera temprana. Los unit tests son poderosos y versátiles y ofrecen una gran variedad de ventajas.

- Permiten simular condiciones de errores, además de abarcar una gran cantidad de código de la aplicación para la realización de testing.
- Incrementan la productividad ya que permiten entregar código de calidad (código que ya ha pasado por el testing) sin tener que esperar a que los demás componentes de la aplicación se encuentren listos.
- Una prueba unitaria que ha sido aprobada, es decir, que ha cumplido con los resultados esperados confirma que el código funciona de manera adecuada, lo que permite agregar nuevas funcionalidades o modificaciones de nuevas características en la aplicación.
- Reduce la necesidad de hacer debug a la aplicación, debido a que especifican de manera concreta el método que está fallando, entre otras ventajas.

4.7.2 Integration Testing (Pruebas de Integración)

El integration testing se basa en el unit testing. Sin embargo, el integration testing va un paso más allá del unit testing. Una vez que se tiene el test para una clase, el siguiente paso es enganchar la clase con otros métodos y servicios. De manera más formal el integration testing es un proceso en el desarrollo de software en el cual varias unidades de trabajo son combinadas y probadas como grupos de múltiples maneras (Tahchiev, Leme, Massol, & Gregory).

Examinar el comportamiento y la interacción entre componentes de la aplicación es el objetivo del integration testing. El integration testing puede exponer problemas con las interfaces entre los distintos componentes que conforman una aplicación antes de que el problema ocurra durante la ejecución de la aplicación en un ambiente de producción.

Aparte de combinar varias unidades de trabajo para la realización del testing, el integration testing se diferencia fundamentalmente del unit testing en el ambiente al que pueden acceder. Es decir, el integration testing puede acceder a una base de datos o lo que sea que asegure que todo el código y los diferentes componentes dentro de la aplicación funcionen de manera correcta.

Para el ambiente de desarrollo de aplicaciones Java existen una gran cantidad de frameworks para la programación de pruebas automatizadas por software. Para el desarrollo de este proyecto se utilizó JUnit y Mockito como se muestra en las secciones siguientes.

4.7.3 JUnit y Mockito

JUnit es un framework simple pero poderoso desarrollado en el año de 1997 para el desempeño

de pruebas automatizadas en el lenguaje de programación Java. Además es una instancia de la arquitectura xUnit. Ha sido liberado bajo la licencia Eclipse Public License versión 1.0.

JUnit es un framework que posee muchas características que hacen fácil y sencillo escribir y correr tests en él.

- Separa las instancias de las clases test y los cargadores de clases para evitar efectos indeseados sobre los tests.
- JUnit provee anotaciones, algunas de ellas para la inicialización de recursos: @Before, @BeforeClass, @After y @AfterClass.
- Cuenta con una variedad de métodos assert para hacer más sencillo el chequeo de resultados de los tests.
- Integración con herramientas populares como Ant y Maven e IDEs populares como Eclipse, NetBeans, IntelliJ y JBuilder.

Para ilustrar el uso de JUnit, a continuación se lista un test muy sencillo tomado del libro JUnit in Action (Tahchiev, Leme, Massol, & Gregory).

```
import static org.junit.Assert.*;
import org.junit.Test;

public class CalculatorTest {

	@Test
	public void testAdd() {
		Calculator calculator = new Calculator();
		double result = calculator.add(10, 50);
		assertEquals(60, result, 0);
	}

}
```

Como puede apreciarse en el listado se trata de un test muy simple.

1. Se define la clase test en este caso con el nombre CalculatorTest. Es una práctica común terminar el nombre de la clase con Test. La clase debe ser pública.
2. Se marca el método testAdd() con la anotación @Test que indica que se trata de un método unit test. Una buena práctica es nombrar los métodos con patrón testXXX. Sin embargo no existen restricciones para el nombramiento de los métodos así como tampoco de las clases.
3. Se crea una instancia de la clase Calculator que es precisamente el objeto que se encuentra bajo el test. Inmediatamente se ejecuta el método a probar de la clase

Calculator, pasándole dos valores conocidos.

4. Para revisar los resultados del test, se llama al método assertEquals(). El método assertEquals() compara los dos valores que le son pasados como parámetros. Comparará el valor 60 contra el valor de la variable result, que si el método de la clase Calculator funciona de manera adecuada, dicha variable debería tener un valor de 60. El tercer parámetro que corresponde al valor 0 se trata de un valor delta, este delta es un rango que se utiliza principalmente cuando se realizan comparaciones con tipos de datos flotantes debido a que se tienen errores de redondeo o de truncamiento, para este caso ese valor es ignorado puesto que los valores pasados como parámetros son valores enteros. Si la prueba no es exitosa, JUnit lanzará una unchecked exception es decir una excepción que hereda de las clases RuntimeException, Error o sus subclases.

Mockito es un framework para facilitar el test automático de programas Java, que se distribuye bajo la licencia MIT, se utiliza en conjunto con JUnit y permite crear pseudo objetos llamados "Mock objects" que permiten abstraer las dependencias de una clase Java y enfocarse solo en probar el código de la clase objetivo. Mockito fue creado por Szczepan Faber y otros, la página oficial de Mockito (Faber, 2012), a continuación un ejemplo del uso de Mockito:

Sea un servicio que recibe como parámetro un entero de tal manera que si el valor del entero es mayor a 5 se tira una excepción, de lo contrario se trae de la base de datos una entidad que se identifica con el valor dado como parámetro:

```
public class MyService {

	@Resource
	private MyRepository myRepository;

	public MyEntity getMyEntity(Integer param) throws Exception {
		if (param > 5) {
			throw new Exception("param should never be greather than 5");
		}

		return myRepository.findMyEntity(param);
	}

}
```

El test usando Mockito y JUnit sería el siguiente:

```
@RunWith(MockitoJUnitRunner.class)
public class MyServiceTest {

	@Mock private MyRepository myRepositoryMock;
```

```
    @InjectMocks private MyService myService;
    private MyEntity expectedMyEntity;

    @Test
    public void testGetMyEntity() {
        Integer param = 5;

        expectedMyEntity = new MyEntity();

        when(myRepositoryMock.findMyEntity(param)).thenReturn(expectedMyEntity);

        MyEntity result = myService.getMyEntity(param);

        Assert.assertEquals(expectedMyEntity, result);

        verify(myRepositoryMock, times(1)).findMyEntity();
    }

    @Test(expected=Exception.class)
    public void testGetMyEntityException() {
        Integer param = 6;

        myService.getMyEntity(param);
    }

}
```

Se marca la clase con la anotación @RunWith(MockitoJUnitRunner.class), inicializa mocks anotados con @Mock y valida el uso del framework después de cada test. Se define un objeto mock con la anotación @Mock, Mockito crea instancia e inyecta el mock, se define una referencia a la entidad *MyEntity* que una vez instanciada será usada para realizar comparaciones. Se define el primer test, en él se declara una variable entera con un valor de 5, se instancia la referencia a la entidad *MyEntity*, se realiza lo que se conoce como "method stub" lo que significa que Mockito le indica que hacer al objeto mock cuando se llama al método findMyEntity() en este caso regresará la instancia de *expectedMyEntity*, después se obtiene un objeto de tipo *MyEntity* llamado *result* directamente del servicio, para finalmente comparar el objeto result contra el objeto *expectedMyEntity*, adicionalmente se puede verificar que el método findMyEntity() se haya llamado solo una vez.

Se define un segundo test que se enfoca en probar otra rama de la ejecución cuando el valor de *param* es mayor a 5 y el método debe tirar una excepción.

La tecnología para el desarrollo de aplicaciones empresariales es un factor muy importante,

debido a que puede facilitar el desarrollo de las mismas, disminuyendo el tiempo de desarrollo y simplificando los componentes que conforman la arquitectura de las aplicaciones por medio de la separación de contenido. La tecnología que se utiliza para desarrollar la aplicación permite fácilmente escalar la aplicación para su uso masivo y además es fácil de desplegar en servidores de hosting o en la nube, las tecnologías usadas son tecnologías en auge para el desarrollo de software.

En el siguiente capítulo se muestra cómo fue diseñada la aplicación, los patrones de diseño empleados, los módulos de los que consta y sus funciones, el flujo de la aplicación y el modelado de los datos.

Capítulo 5 Diseño de la aplicación

En este capítulo se muestra todo lo concerniente al diseño, flujo, componentes, patrones de diseño y servicios web de la aplicación. Los módulos de los que consta, cómo funcionan estos módulos, como fueron diseñados, etcétera. Se mostrarán las definiciones de los diferentes servicios web desarrollados que sirven como interfaz a los módulos de la aplicación y las funciones de los mismos.

5.1 Patrones de diseño

"Un patrón de diseño es la mejor práctica documentada o el núcleo de una solución que ha sido aplicada en múltiples ambientes para resolver un problema que recurre en un conjunto específico de situaciones", (Kuchana, 2004).

Para el desarrollo de este proyecto de tesis se utilizaron varios patrones de diseño, los cuales serán explicados en las siguientes secciones.

5.1.1 Arquitectura MVC

La arquitectura MVC (Modelo Vista Controlador) es un patrón de diseño formulado en el año de 1979 por Trygve Reenskaug. El objetivo del patrón de diseño MVC es aislar la lógica de negocio de la interfaz de usuario. Por medio del uso del MVC, se logra una separación de contenido en varias capas como lo muestra la Figura 5.1.

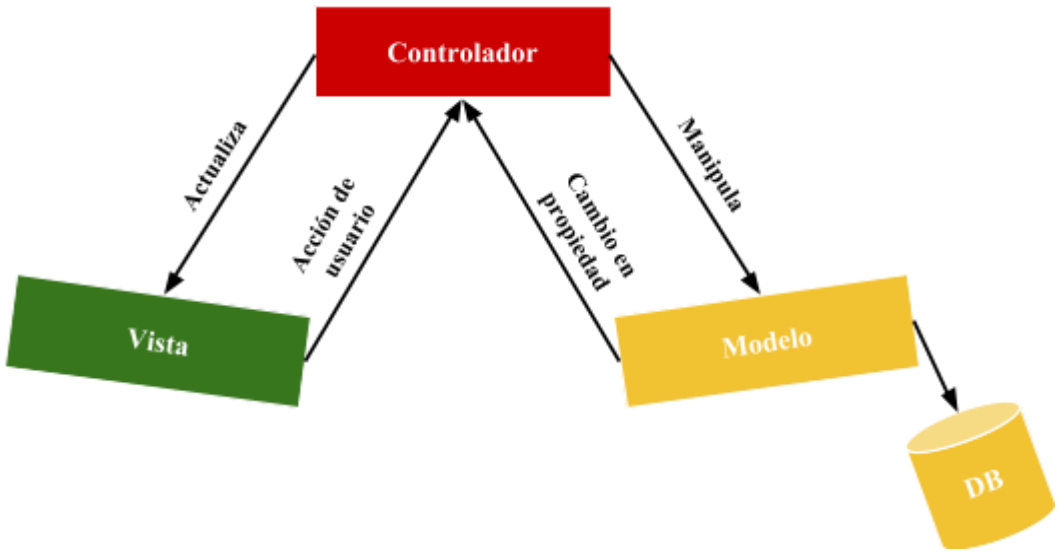


Figura 5.1 Diagrama general del patrón de diseño MVC.

- La capa del modelo. Representa la información o datos de la aplicación y las reglas de negocio para la manipulación de dicha información. Es la única parte que tiene acceso o comunicación directa a la base de datos.
- La capa de la vista. Es la parte responsable de la presentación, es decir, como se muestra la información al usuario. Esta capa accede a esa información o estado del modelo por medio del controlador.
- La capa del controlador se encarga de manejar las acciones del usuario y actualizar el estado del modelo en base a esas acciones, además de hacer accesible ese estado a la vista.

5.1.2 Patrón de diseño DAO

El patrón de diseño DAO consiste en agregar una interfaz hacia la base de datos por medio de componentes reusables y que al ser modificados no alteran otros módulos de la aplicación, de forma que dicha interfaz esconde todos los detalles al resto de la aplicación que se refieren al almacenamiento de la información.

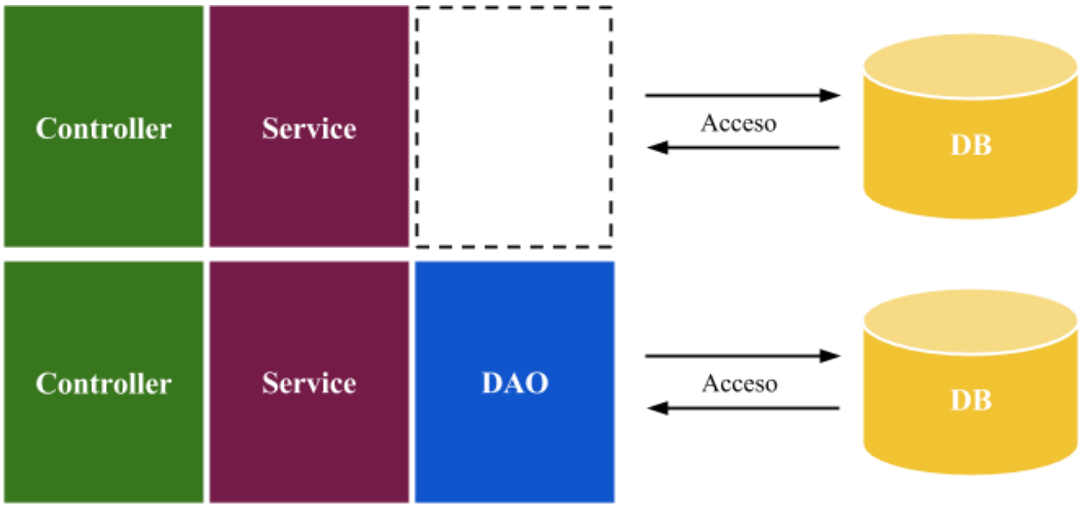


Figura 5.2 Diagrama del patrón de diseño DAO.

5.1.3 Patrón de diseño Singleton

“El Patrón de diseño Singleton asegura que una clase tenga solo una instancia, y provee un punto de acceso global a dicha instancia”, (Freeman, Freeman, Sierra, & Bates, 2004).

El patrón de diseño Singleton restringe la instanciación de una clase a un solo objeto. Previene que cualquier otra clase pueda crear una instancia de sí misma. Para poder obtener una instancia

se debe de hacer por medio de la clase. Provee un punto de acceso global a la clase y dependiendo de su implementación el singleton puede ser implementado de manera “lazy”, lo cual es particularmente importante para los objetos de uso intensivo de recursos. El término “Lazy” se refiere a que los objetos son creados hasta el momento en que la aplicación los llame o utilice por primera vez. Este patrón de diseño es utilizado en la aplicación web en la inyección de dependencias por eficiencia para reutilizar los objetos inyectados.

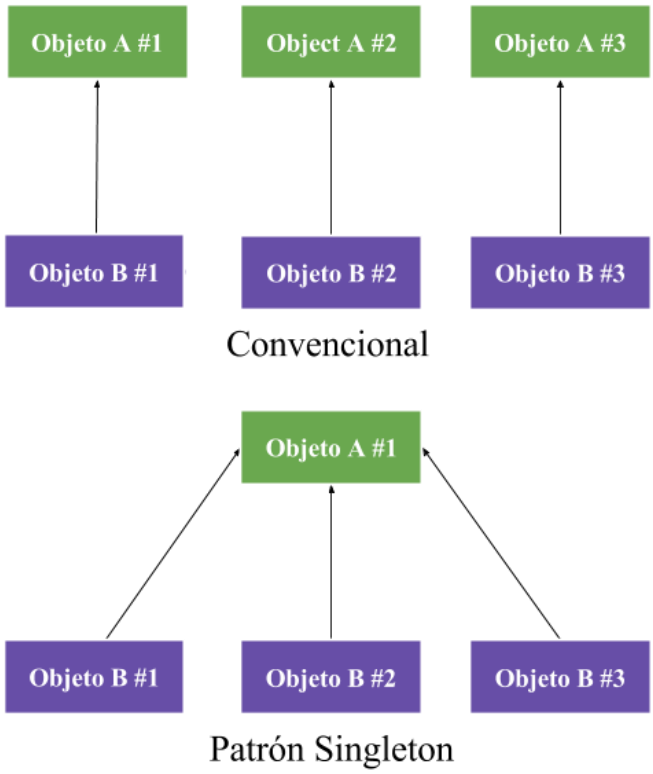


Figura 5.3 Diagrama del patrón de diseño Singleton.

En la sección del modelo de la aplicación se verá cómo se utilizan estos patrones de diseño desde un punto de vista más técnico en la aplicación desarrollada.

5.2 Módulos de la aplicación

La aplicación se encuentra estructurada en una serie de módulos, cada uno realiza una función específica. El funcionamiento de cada uno de los módulos debe de ser independiente, es decir, ninguno de los módulos implementados debe interferir en el funcionamiento de alguno de los demás módulos de la aplicación.

El diseño modular en las aplicaciones es una técnica del diseño de software que permite la

separación de la funcionalidad de una aplicación, de tal manera que cada una de estas separaciones o módulos representa una sección de código independiente a los demás módulos de la aplicación, de esta forma es posible mejorar la mantenibilidad de la aplicación.

La Figura 5.4 muestra los diferentes módulos que componen a la aplicación y la interfaz hacia ellos a través de un REST API.

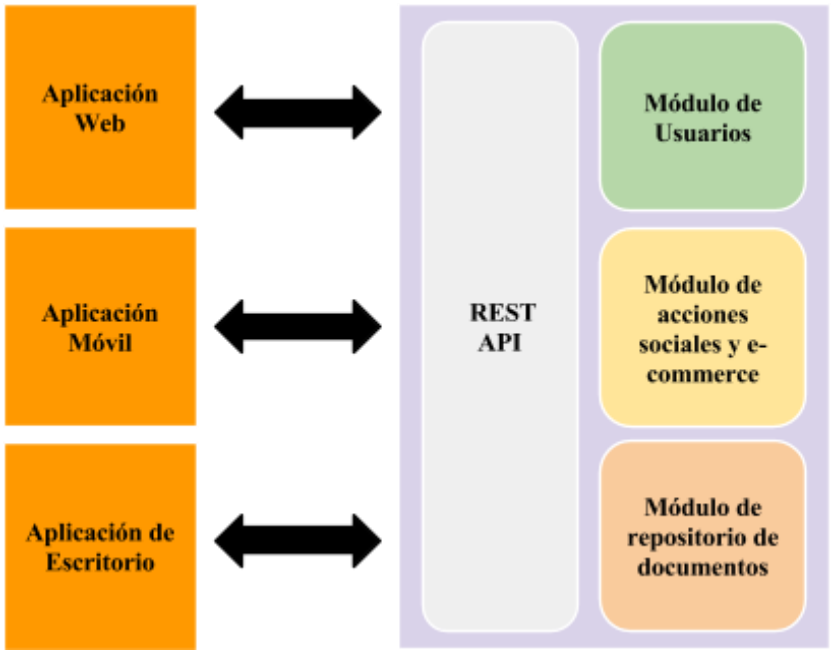


Figura 5.4 Diagrama de los módulos que componen la aplicación y el acceso a la misma por medio de un REST API.

Existen tres módulos que conforman el núcleo básico de servicios para esta aplicación los cuales son los siguientes:

1. Módulo de usuarios
2. Módulo de acciones sociales y e-commerce
3. Módulo de repositorio de documentos

Las diferentes funciones que proveen los módulos son accedidas por medio de un REST API, brindando así la independencia de plataforma, que, como es posible observar en la Figura 5.4 el REST API puede ser consumido por cualquier tipo de aplicación.

5.2.1 Módulo de usuarios

El módulo de usuarios es el módulo fundamental de la aplicación. En este módulo se manejan

varios aspectos primordiales de la aplicación, a continuación se listan algunos de ellos.

1. **Usuarios**. Permite dar de alta nuevos usuarios de la aplicación, así como su activación y las demás operaciones para la administración de la información concerniente al usuario. Cuando los usuarios son dados de alta dentro del sistema lo hacen con un estatus de "Nuevo", es por eso que es necesaria una posterior activación de la cuenta.
2. **Perfiles**. Permite dar de alta perfiles vinculados al usuario, activación y demás operaciones para la administración de la información concerniente a los perfiles de usuario. Existen dos clases de perfiles: comercializadores y productores. Un usuario puede tener dos perfiles vinculados a su cuenta como máximo: un perfil de tipo comercializador y otro de tipo productor si así lo requiriese.
3. **Productos**. Permite dar de alta productos y vincularlos a sus perfiles así como todas las operaciones para la administración de la información correspondiente a los productos y asociaciones entre productos y perfiles. Como productor o comercializador el usuario puede dar de alta los productos que produzca o con los que lucre con el motivo de poder ofrecerlo a los diferentes usuarios de la plataforma que estén interesados en dichos productos.

Las operaciones realizadas en el módulo de usuarios se muestran en la siguiente tabla. Las operaciones listadas son operaciones generales, no todas las funciones del módulo se encuentran listadas en la Tabla 5.1, puesto que existen operaciones subyacentes. Para ver una lista completa de los servicios que provee el módulo se debe ir a los apéndices al final de este documento.

Tabla 5.1 Operaciones realizadas por el Módulo de usuarios.

Entity	Servicio	Descripción
Usuario	Dar de alta usuario	Creación de un usuario dentro del sistema
Usuario	Desactivar usuario	El usuario puede desactivar su cuenta de usuario, una vez que éste haya sido dado de alta
Usuario	Actualizar usuario	El usuario puede actualizar la información de su usuario dado de alta en el sistema
Usuario	Actualizar contraseña	El usuario puede actualizar la contraseña de usuario.
Usuario	Resetear contraseña	El usuario puede resetear su contraseña si por alguna razón la olvidó por ejemplo
Usuario	Activar usuario	El usuario puede activar su cuenta, cuando el usuario es creado o dado de alta en el sistema lo hace con un estatus que le prohibe realizar

		cualquier acción hasta que se active la cuenta por medio de esta función
Usuario	Leer información de usuario	El usuario puede leer información sobre otros usuarios
Perfil	Dar de alta perfil	El usuario puede dar de alta un perfil (crear) que puede ser de dos tipos: comercializador o productor
Perfil	Desactivar perfil	El usuario puede desactivar cualquier perfil que tenga dado de alta
Perfil	Activar perfil	El usuario puede activar un perfil que haya sido dado de baja
Perfil	Actualizar perfil	El usuario puede actualizar la información de su perfil
Perfil	Leer información de perfil	El usuario puede ver perfiles de otros usuarios
Producto	Crear producto	El usuario puede dar de alta un producto
Producto	Activar producto	El usuario puede activar un producto que haya sido dado de baja
Producto	Desactivar producto	El usuario puede desactivar un producto que esté dado de alta y activado
Producto	Actualizar producto	El usuario puede actualizar información de sus productos
Producto	Leer información de producto	El usuario puede leer información de productos
Producto	Asociar producto a perfil	El usuario puede asociar un producto a alguno de sus perfiles
Producto	Actualizar asociación entre perfil y producto	El usuario puede actualizar la información existente entre la asociación de alguno de sus perfiles y un producto
Producto	Desactivar asociación entre perfil y producto	El usuario puede desactivar la asociación existente entre alguno de sus perfiles y un producto
Producto	Leer asociación entre perfil y producto	El usuario puede leer las demás asociaciones entre perfiles de otros usuarios y productos
Producto	Activar asociación entre	El usuario puede activar una asociación entre

	perfil y producto	alguno de sus perfiles y un producto que haya sido dada de baja con anterioridad

5.2.2 Módulo de acciones sociales y E-commerce

El módulo de acciones sociales provee la interacción social entre los usuarios de la plataforma. Dicha interacción social se da por medio de las diferentes características existentes como la publicación de contenido en el pinboard (o muro) del usuario, el seguimiento y conexión entre usuarios, etc. A continuación se listan tales acciones sociales con más detalle.

1. Un usuario puede seguir a otro y ser seguido a su vez por otros, de manera que, si el usuario A sigue al usuario B, A puede ver todas las publicaciones del pinboard de B y hacer comentarios en ellas. De la misma manera, si A tiene seguidores, éstos podrán ver el contenido que éste publica y comentar dicho contenido.
2. El usuario puede hacer solicitudes de conexión con otros usuarios con la finalidad de hacer negocios con ellos, si la solicitud es aceptada:

    a. Puede solicitar cotizaciones de productos dados de alta por su contacto que sean de su interés.
    b. Puede enviar cotizaciones de productos que le hayan sido solicitadas por sus contactos.
    c. Puede enviar mensajes a sus contactos.

5.2.2.1 Cotizaciones

Las cotizaciones son un medio por el cual se desarrollan las negociaciones, informan sobre el valor de productos o servicios (en este caso productos agrícolas). Para el desarrollo de este proyecto de Tesis las cotizaciones son una parte fundamental puesto que representan el inicio de las negociaciones entre dos usuarios y también el inicio o primer medio de contacto entre productores y comercializadores para realizar importación/exportación de productos.

Las cotizaciones y solicitudes de cotización de la aplicación están basadas en dos estándares: el estándar cXML versión 1.2.024 y el estándar openTRANS versión 2.1.

cXML

cXML (Commerce eXtensible Markup Language) es un estándar creado por la empresa Ariba (Ariba, Inc., 2014) en 1999 para la comunicación de información de comercio electrónico. Es un lenguaje versátil y abierto para los requerimientos de transacciones de:

- Centros de comercio electrónico en red
- Catálogos de productos electrónicos
- Catálogos “PunchOut”
- Aplicaciones de adquisiciones
- Compradores
- Proveedores
- Proveedores de servicios de comercio electrónico

Los documentos que provee el estándar son los siguientes:

- Documentos cXML básicos
- Confirmation and Ship Notice (Confirmación y notificaciones de envío)
- Invoice (Facturas)
- Type Definition (Definición de tipos)
- Payment Remittance (Remesas de pago)

cXML define una solicitud de cotización como lo muestra la siguiente estructura XML:

```
<QuoteRequest>
        <QuoteRequestHeader>
                header information
        </QuoteRequestHeader>
        <QuoteItemOut>
                QuoteItemOut information
        </QuoteItemOut>
</QuoteRequest>
```

1. El QuoteRequestHeader almacena información acerca de los detalles de la solicitud de cotización enviada a un proveedor.
2. El QuoteItemOut almacena detalles sobre los items enviados en la solicitud.

En cXML la manera de responder a una solicitud de cotización es por medio de la estructura QuoteMessage, a continuación se muestra la estructura en XML:

```
<QuoteMessage>
        <QuoteMessageHeader>
                header information
        </QuoteMessageHeader>
        <QuoteItemIn>
                QuoteItemIn information
        </QuoteItemIn>
</QuoteMessage>
```

1. El elemento QuoteMessageHeader almacena los detalles del encabezado del QuoteMessage que es enviado al comprador.
2. El elemento QuoteItemIn detalla información sobre los items.

Véase la página oficial del estándar cXML (cXML.org, s.f.).

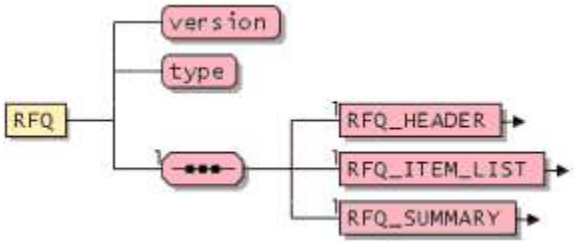
openTRANS

openTRANS es un estándar basado en XML para el intercambio electrónico de documentos de negocios (Transacciones). Es una iniciativa con socios industriales alemanes e internacionales bajo el liderazgo de Fraunhofer IAO y la Universidad de Duisburg-Essen. El estándar especifica los siguientes documentos:

- RFQ (Request For Quotation o Solicitud de Cotización)
- QUOTATION (Cotización)
- ORDER (Orden)
- ORDERCHANGE (Cambio de Orden)
- ORDERRESPONSE (Repuesta de Orden)
- DISPATCHNOTIFICATION (Notificación de envío o despacho)
- RECEIPTACKNOWLEDGEMENT (Acuse de recibo)
- INVOICE (Factura)
- INVOICELIST (Lista de facturas)
- REMITTANCEADVICE (Aviso de pago)

openTRANS define una estructura muy amplia en el caso de las solicitudes de cotización y cotizaciones, para fines prácticos solo se muestra la estructura básica:

Figura 5.5 Estructura general de una solicitud de cotización en el estándar openTRANS.



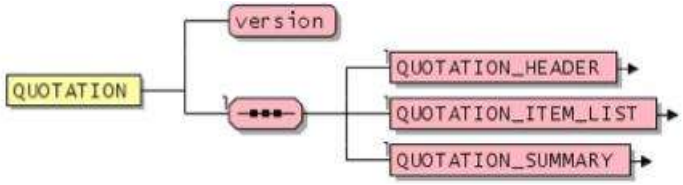
Fuente: (Schmitz, Kelkar, Otto, & Weiner , 2009).

Cada documento RFQ válido de negocio en el formato openTRANS comienza por el elemento RFQ que consiste de un encabezado (RFQ_HEADER), un elemento a nivel ítem (RFQ_ITEM_LIST) y un resumen (RFQ_SUMMARY).

1. El encabezado es el comienzo del documento de negocio y contiene información global como información sobre proveedores tal como acuerdos entre compradores y proveedores, además de otros.
2. El elemento a nivel de item detalla los elementos que conforman a la solicitud de cotización que para nuestro caso son productos agrícolas.
3. Finalmente el resumen contiene un compendio de la información de la solicitud de cotización que puede ser usada con fines de control o estadísticos.

Para el caso de las cotizaciones openTRANS define la estructura QUOTATION de la siguiente manera:

Figura 5.6 Estructura general de una cotización en el estandar openTRANS.



Fuente: (Schmitz, Kelkar, Otto, & Weiner , 2009).

Al igual que la estructura RFQ la estructura QUOTATION también consta de tres partes: un encabezado una lista de items y un resumen que básicamente detallan la misma información de una estructura RFQ.

Véase la página oficial del estándar openTRANS (openTRANS, 2011).

Entidades Cotización y Solicitud de Cotización

Debido a la gran cantidad de atributos y elementos que detallan los estándares anteriores solo se listarán los atributos sobre los cuales las entidades Cotización (Quotation) y Solicitud de Cotización (QuotationRequest) que se utilizan en este trabajo están basadas.

Elementos tomados del estándar cXML para el diseño de la entidad Quotation:

1. QuoteId
2. QuoteDate
3. Total
4. Contact
5. Comments
6. Quantity

7. ItemId
8. Contact
9. Shipping

Elementos tomados del estándar openTRANS para el diseño de la entidad Quotation:

1. DELIVERY_DATE
2. LANGUAGE
3. CURRENCY
4. PAYMENT
5. ORDER_UNIT
6. TRANSPORT

La combinación (unión) de los elementos obtenidos de los estándares cXML y openTRANS listados anteriormente dan como resultado los atributos que componen a la entidad Quotation que se derivó de este trabajo y que muestra la Figura 5.7.

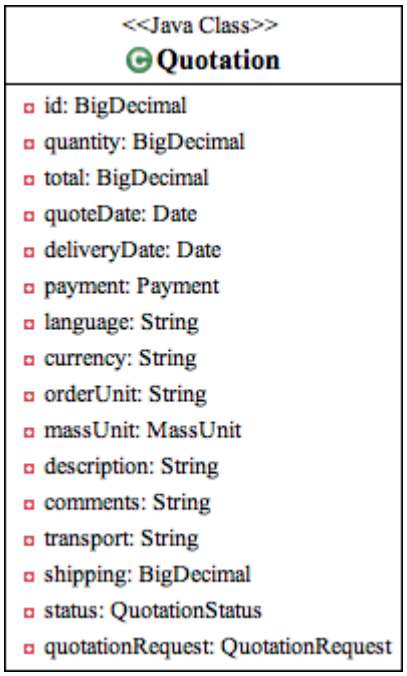


Figura 5.7 Entidad Quotation.

1. **id** (QuoteId). Id del Quotation.
2. **quantity** (Quantity). Cantidad total de productos que se cotiza
3. **total** (Total). Monto total de la cotización.
4. **quoteDate** (QuoteDate). Fecha en que se realiza la cotización.
5. **deliveryDate** (DELIVERY_DATE). Fecha de entrega de producto.

6. **payment** (PAYMENT). Modo de pago.
7. **language** (LANGUAGE). Lenguaje de la cotización.
8. **currency** (CURRENCY). Moneda sobre la cual están establecidos los montos de la cotización.
9. **orderUnit** (ORDER_UNIT). Unidad de medida de los pedidos a cotizar.
10. **description**. Este campo no se corresponde con los estándares, pero es una descripción de la cotización.
11. **comments** (Comments). Comentarios acerca de la cotización.
12. **transport** (TRANSPORT). Medio de transporte.
13. **shipping** (Shipping). Costo de envío total.
14. **status**. Estatus de la cotización.
15. **quotationRequest** (Contact, ItemId). Referencia a la solicitud de cotización en donde también se especifica el contacto al cual se le envía la cotización y la información de los items.
16. **massUnit**. Especifica las unidades para la cantidad de producto.

Elementos tomados del estándar cXML para el diseño de la entidad QuotationRequest:

1. RequestId
2. RequestDate
3. OpenDate
4. CloseDate
5. Description
6. Contact
7. ShipTo
8. Quantity
9. ItemId
10. Shipping
11. Comments

Elementos tomados del estándar openTRANS para el diseño de la entidad QuotationRequest:

1. DELIVERY_START_DATE
2. DELIVERY_END_DATE
3. LANGUAGE
4. CURRENCY
5. TRANSPORT
6. ORDER_UNIT
7. DELIVERY_DATE

La combinación (unión) de los elementos de ambos estándares mostrados en los listados anteriores dan como resultado los atributos que componen a la entidad QuotationRequest que se usará en este trabajo y que se muestra en la Figura 5.8.

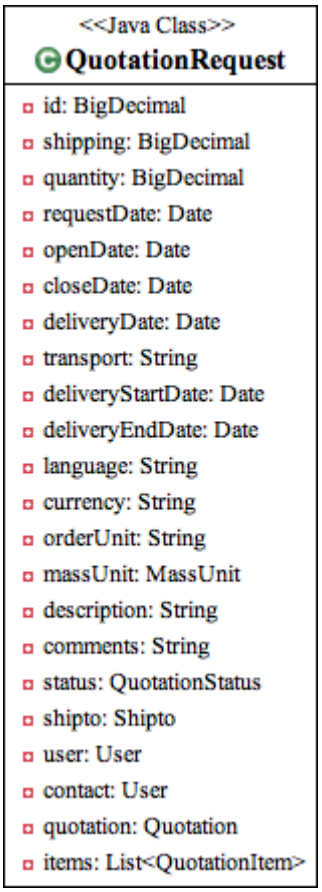


Figura 5.8 Entidad QuotationRequest.

1. **id** (RequestId). Id del QuotationRequest.
2. **shipping** (Shipping). Costo de envío total.
3. **quantity** (Quantity) Cantidad de producto.
4. **requestDate** (RequestDate). Fecha en que se hizo la solicitud de cotización.
5. **openDate** (OpenDate). Fecha de inicio para responder la solictud.
6. **closeDate** (CloseDate). Fecha de fin para responder a la solicitud.
7. **deliveryDate** (DELIVERY_DATE). Fecha de entrega del producto.
8. **transport** (TRANSPORT). Medio de transporte.
9. **deliveryStartDate** (DELIVERY_START_DATE). Fecha de inicio en la que el pedido puede llegar.
10. **deliveryEndDate** (DELIVERY_END_DATE). Fecha límite a la que el pedido puede llegar.
11. **language** (LANGUAGE). Lenguaje de la solicitud.

12. **currency** (CURRENCY). Moneda de los montos.
13. **orderUnit** (ORDER_UNIT). Unidad de medida de orden de los productos solicitados.
14. **description** (Description). Descripción sobre la cotización.
15. **comments** (Comments). Comentarios sobre la cotización.
16. **status**. Estatus de la cotización.
17. **shipto** (Shipto). Información de envío de la mercancía.
18. **user**. Usuario que envía la solicitud.
19. **contact** (Contact). Usuario al que se le envía la solicitud.
20. **quotation**. Cotización que representa la respuesta a esta solicitud.
21. **items** (ItemId). Lista de items solicitados para su cotización.
22. **massUnit**. Especifica las unidades para la cantidad de producto.

Para más detalles sobre las cotizaciones véase la Sección 5.3.3 Modelo de Datos en la cual se muestran las asociaciones entre entre la entidad Quotation y QuotationRequest con las demás entidades.

A continuación la Tabla 5.2 detalla las operaciones del Módulo 2.

Tabla 5.2 Operaciones realizadas por la entidad Usuario en Módulo de acciones sociales y e-commerce.

Servicio	Descripción
Enviar mensaje	Un usuario puede mandar un mensaje privado a otro usuario
Consultar mensajes	Un usuario puede recibir mensajes privados de otros usuarios
Borrar mensajes	El usuario puede borrar los mensajes que le han sido enviados por otros usuarios
Crear posts	El usuario puede publicar posts en su pinboard
Borrar posts	El usuario puede borrar los posts que haya publicado en su pinboard aunque éstos contengan comentarios de otros usuarios o de su misma autoría
Editar Post	El usuario puede editar los post que haya publicado anteriormente
Leer posts	El usuario puede leer los posts de sus contactos y los de él mismo junto con todos los comentarios de dicho post
Leer posts de un usuario al que se está siguiendo	El usuario puede leer todos los posts de un usuario al que sigue en ese momento
Borrar comentario de post	El usuario puede borrar comentarios de su propio

	pinboardPost
Comentar posts de usuarios	El usuario puede comentar los posts de otros usuarios o los de él mismo
Borrar comentario	El usuario puede borrar comentarios hechos por él
Seguir contacto	El usuario puede seguir a otros usuarios de la red social y ver las publicaciones que éstos compartan
Obtener contactos que sigue el usuario	Traer todos los contactos que el usuario sigue.
Obtener contactos que siguen al usuario	Traer todos los contactos que están siguiendo al usuario
Dejar de seguir contacto	El usuario puede dejar de seguir a un contacto que ya esté siguiendo, dejando así de recibir notificaciones sobre la actividad de éste
Conectar con otro usuario	El usuario puede hacer conexión con otros usuarios con la finalidad de hacer negocio con este usuario
Borrar la conexión con usuario	El usuario puede desconectarse de los usuarios con los que ya tenga una conexión.
Obtener contactos con los que se tienen conexiones	El usuario puede ver todos los usuarios con los que ha hecho conexión
Obtener solicitudes de conexión	El usuario puede ver todas las solicitudes de conexión que otros usuarios le han hecho
Aceptar solicitud de conexión	El usuario puede aceptar solicitudes de conexión de otros usuarios
Denegar solicitud de conexión	El usuario puede denegar solicitudes de conexión de otros usuarios o terminar la conexión que tenga con otro usuario
Enviar solicitud de cotización	El usuario puede enviar solicitudes de cotización a cualquiera de sus contactos sobre los productos que a éste le interesen
Guardar draft de una solicitud de cotización	El usuario puede guardar drafts de las solicitudes de cotización que pretenda enviar a cualquiera de sus contactos
Actualizar una solicitud de cotización	El usuario puede actualizar la información de las solicitudes de cotización que haya enviado a sus contactos, siempre y cuando éstas no hayan sido respondidas mediante el envío de la cotización por parte de los contactos

Borrar solicitudes de cotización	El usuario puede borrar solicitudes de cotización, hacer ésto significa que el status de la solicitud será cambiado a "Borrado" al igual que el de la cotización, si es que ya fue respondida la solicitud
Clonar una solicitud de cotización	El usuario puede clonar o copiar algunas de las solicitudes de cotización que haya hecho
Leer solicitudes de cotización	El usuario puede leer la información de las solicitudes de cotización
Enviar cotizaciones	El usuario puede responder a una solicitud de cotización por medio de una cotización
Leer cotizaciones	El usuario puede leer información sobre las cotizaciones
Borrar cotizaciones	El usuario puede borrar cotizaciones, si una cotización es borrada también lo será su solicitud
Guardar borradores de cotizaciones	El usuario puede guardar drafts de las cotizaciones

5.2.3 Módulo de repositorio de documentos

El módulo de repositorio de documentos permite subir, descargar y administrar archivos, ya sean videos, imágenes o documentos. La intención es que el usuario pueda guardar imágenes de perfil, de portada, videos y documentos relacionados al negocio agrícola como certificaciones, facturas, etc.

Las operaciones desempeñadas por el módulo son las mostradas en la Tabla 5.3.

Tabla 5.3 Operaciones realizadas por la entidad Usuario en el módulo del repositorio de documentos.

Servicio	Descripción
Subir archivo	El usuario puede subir archivos al repositorio
Descargar archivo	El usuario puede descargar archivos de manera individual
Leer metadatos de archivo	El usuario puede leer metadatos de un archivo
Leer metadatos de varios archivos	El usuario puede leer los metadatos de varios archivos
Actualizar archivo	El usuario puede actualizar un archivo

Borrar archivo	El usuario puede borrar un archivo
Subir imagen de perfil	El usuario puede subir una imagen para su perfil
Ver imagen de perfil	El usuario puede ver la imagen de su perfil
Actualizar imagen de perfil	El usuario puede actualizar la imagen de su perfil
Subir imagen de portada	El usuario puede subir una imagen de portada a su perfil
Borrar imagen de portada	El usuario puede borrar su imagen de portada
Actualizar imagen de portada	El usuario puede actualizar la imagen de portada
Subir video de perfil	El usuario puede subir un video a su perfil
Borrar video de perfil	El usuario puede borrar su video de perfil
Actualizar video de perfil	El usuario puede actualizar su video de perfil

5.3 Flujo y Modelo de Datos

5.3.1 Flujo de la Aplicación

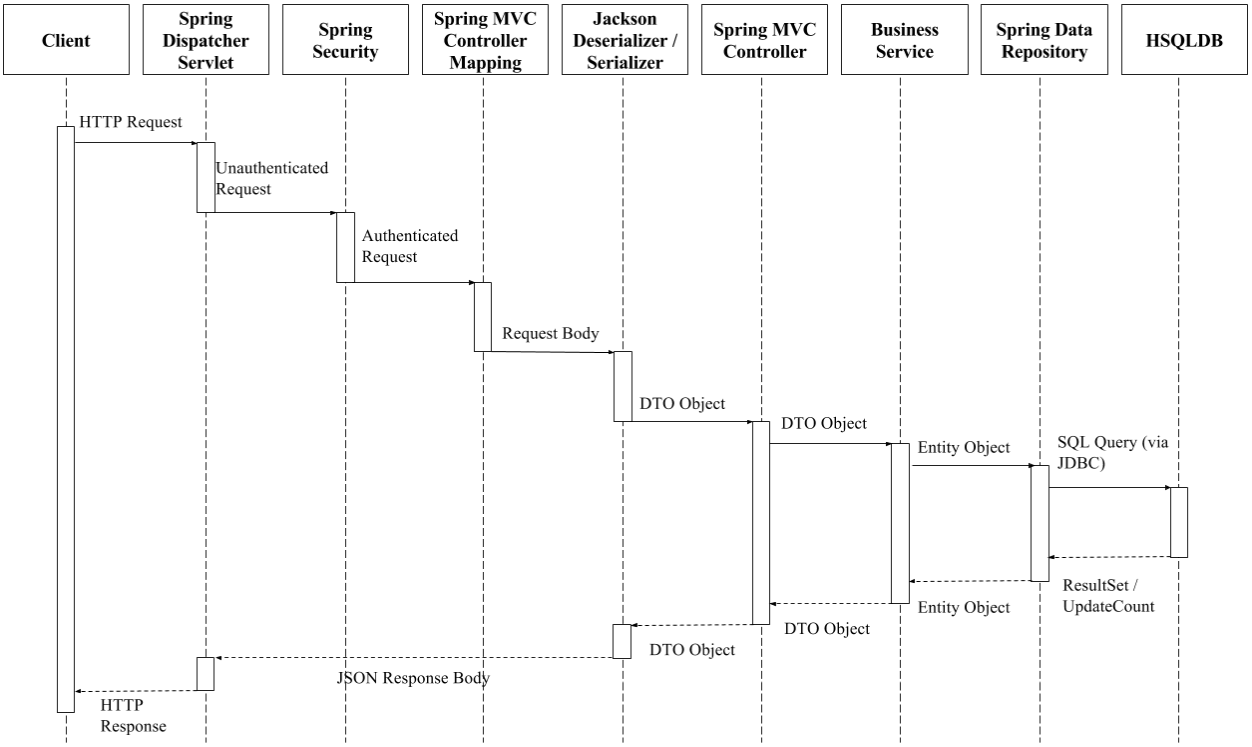


Figura 5.9 Diagrama de secuencia que muestra una descripción a grandes rasgos de la secuencia que sigue una

request típica exitosa de la aplicación desarrollada.

El flujo de la aplicación como se describe de manera visual en la Figura 5.9 se da de la siguiente manera:

1. El cliente manda una petición HTTP (HTTP Request) al servidor que puede contener un objeto JSON o un conjunto de parámetros dentro de la URL.
2. El Spring DispatcherServlet intercepta la petición HTTP que viene del cliente y la redirecciona como una petición no autenticada (Unauthenticated Request) a la capa de Spring Security.
3. Spring Security determina si el usuario ha sido autenticado, es decir, que el usuario y contraseña sean correctos y si lo es manda la petición como una petición autenticada (Authenticated Request) al Spring MVC Controller Mapping.
4. El Spring MVC Controller Mapping mapea la petición del cliente al controlador adecuado (Controller) y le envía el cuerpo de la petición (Request Body).
5. Los datos adjuntos en el cuerpo de la petición son traducidos a objetos Java. En el caso de que el objeto de entrada sea JSON es traducido a un objeto Java DTO (Data Transfer Object). Todo lo anterior es hecho por Jackson.
6. Una vez que se tienen los objetos Java traducidos el controlador (Controller) procesa el cuerpo de la solicitud que contiene los DTOs. Llama al servicio (Service) correspondiente y envía como parámetro el DTO.
7. El servicio desempeña las operaciones CRUD (ver apéndices para la lista completa de servicios REST) de los módulos de la aplicación. Los servicios internamente utilizan a los repositorios (Repository) para acceder a las Bases de Datos. Debido a que los repositorios trabajan con objetos a nivel entidad (Entity Object), los DTOs deben ser mapeados a entidades (Entity Objects) dentro de los servicios, ésto se hace por medio de servicios de mapeo DTO-entidad que también son parte de la capa de servicios.
8. Los repositorios son implementados internamente con objetos DAO que crea Spring, dichos objetos utilizan Hibernate para comunicarse con la BD mediante consultas SQL autogeneradas.
9. Las consultas SQL son procesadas por el manejador de la base de datos.
10. El manejador de la base de datos regresa un “ResultSet” o un “UpdateCount” dependiendo de la operación.
11. La información relacional obtenida de la base de datos es traducida a objetos entidad (Entity) por Hibernate y devuelta al servicio.
12. En el servicio se mapean las entidades o entidad que fueron recuperadas de la base de datos a DTOs.
13. El controlador devuelve el DTO como parte del cuerpo de la respuesta (Response Body) al Spring DispatcherServlet.
14. El cuerpo de la respuesta se traduce a JSON, de nuevo por medio de Jackson.

15. Finalmente el Spring DispatcherServlet regresa al cliente la respuesta HTTP (HTTP Response).

La aplicación se encuentra integrada por una serie de componentes que cumplen una función específica, los cuales se muestran en la Figura 5.10.

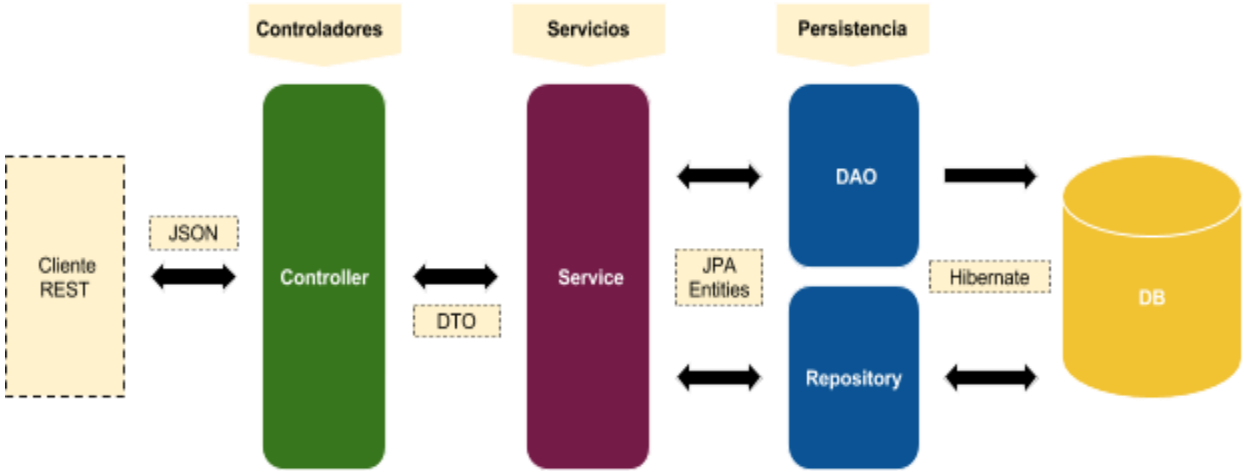


Figura 5.10 Diagrama de componentes de la aplicación, exceptuando módulo de repositorio de documentos.

Como se puede apreciar en la Figura 5.10 la aplicación se encuentra formada por una serie de capas: La capa de controladores, la capa de servicios y la capa de persistencia.

La Figura 5.11 muestra cómo los componentes de la aplicación mostrados en la Figura 5.10 se sitúan dentro del modelo MVC.

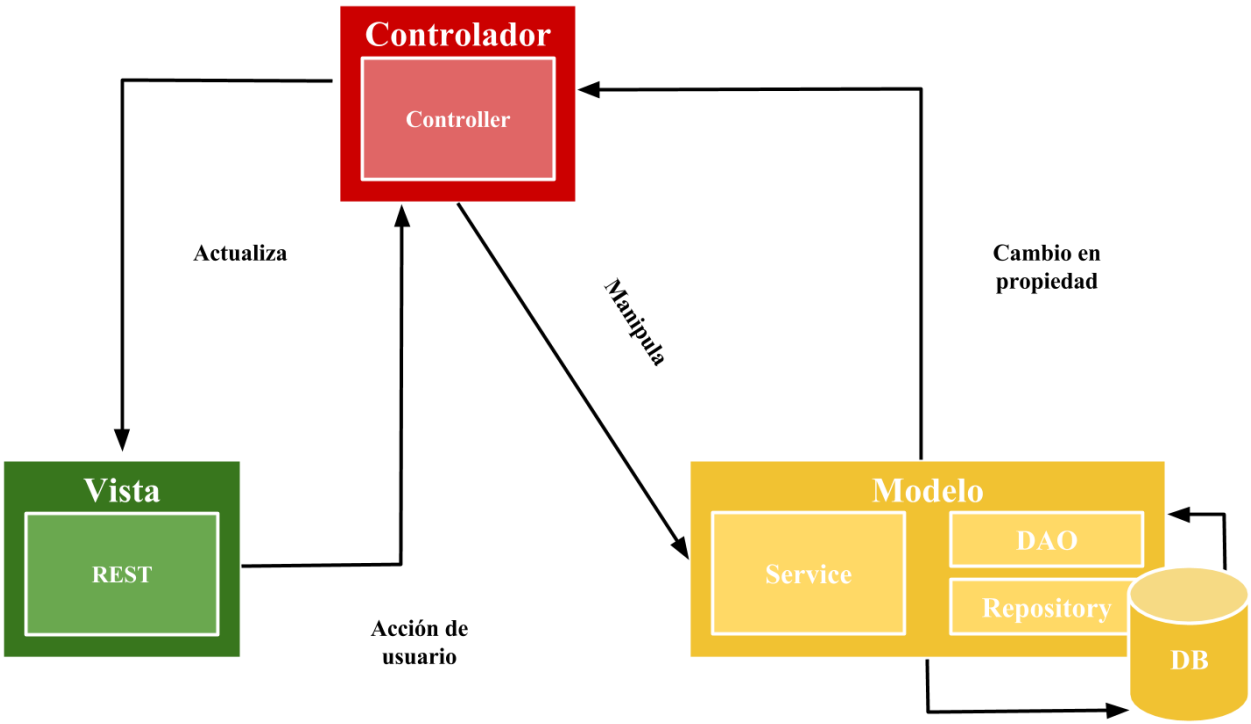


Figura 5.11 Diagrama de los componentes de la aplicación desde el punto de vista del modelo MVC.

La capa de la Vista es el REST API, de forma que, una llamada a alguno de los servicios web que conforman el REST API representa una acción del usuario, la capa del Controlador que está compuesta de componentes Controller debe procesar esa acción por medio de la manipulación de la capa del Modelo. La capa del Modelo es un poco más compleja que las otras dos capas debido a que está compuesta de un mayor número de componentes, los cuales son: Service, DAO y Repository, además, esta capa accesa directamente la información de la base de datos y contiene la lógica de negocio de la aplicación. El componente en la capa del controlador recupera el cambio hecho en la capa del Modelo debido a la acción del usuario y actualiza la capa de la Vista con los cambios recuperados de la capa del Modelo.

5.3.2 Mensajes de error

Cuando una petición es procesada de manera correcta por el servicio web se envía un status 200 en la respuesta HTTP al cliente, de esa forma el cliente puede esperar a que dentro de la respuesta HTTP viaje un objeto JSON, que después el cliente procesará a su conveniencia. Para el caso de errores o condiciones anormales durante el procesamiento de la petición, la aplicación cuenta con un mecanismo de manejo de excepciones, cuando ocurre la excepción ésta se convierte a un objeto ErrorResponse (para informar sobre el error), el cual retorna en representación JSON y se adjunta en el HTTP response, estas respuestas tienen una estructura ordenada que se describe a continuación:

- Contienen un código de status que corresponde con el status HTTP de la respuesta HTTP.
- Contienen un código interno que especifica la capa de la aplicación en la que sucedió el error.
- Tienen una lista de mensajes lanzados por la aplicación.
- Contiene un mensaje para la descripción técnica del error.
- Por último contiene un campo más para mostrar mostrar el URL hacia la documentación del REST API.

A continuación la representación JSON de un ErrorResponse (se corta parte del mensaje técnico por brevedad):


```
{
  "status": 500,
  "messages": [   {
        "message": "There was an error while accessing database",
        "code": "30001"
  }],
  "technicalMessage": "com.snl.lemon.exception.ServiceException: BaseException: There was an error while accessing database at
com.snl.lemon.service.ProductService.updateFarmerProduct(ProductService.java:360) at
...,
  "moreInfo": "<Dirección de documentación del REST API>"
}
```


En el Apéndice A Errores manejados por la aplicación se muestra un listado de todos los errores que se manejan dentro de la aplicación.

5.3.3 Modelo de Datos

En esta sección se presenta el modelo de datos de la aplicación: diagramas de clases del Módulo 1 y 2 y el diagrama E-R de la base de datos que incluye a los Módulos 1 y 2.

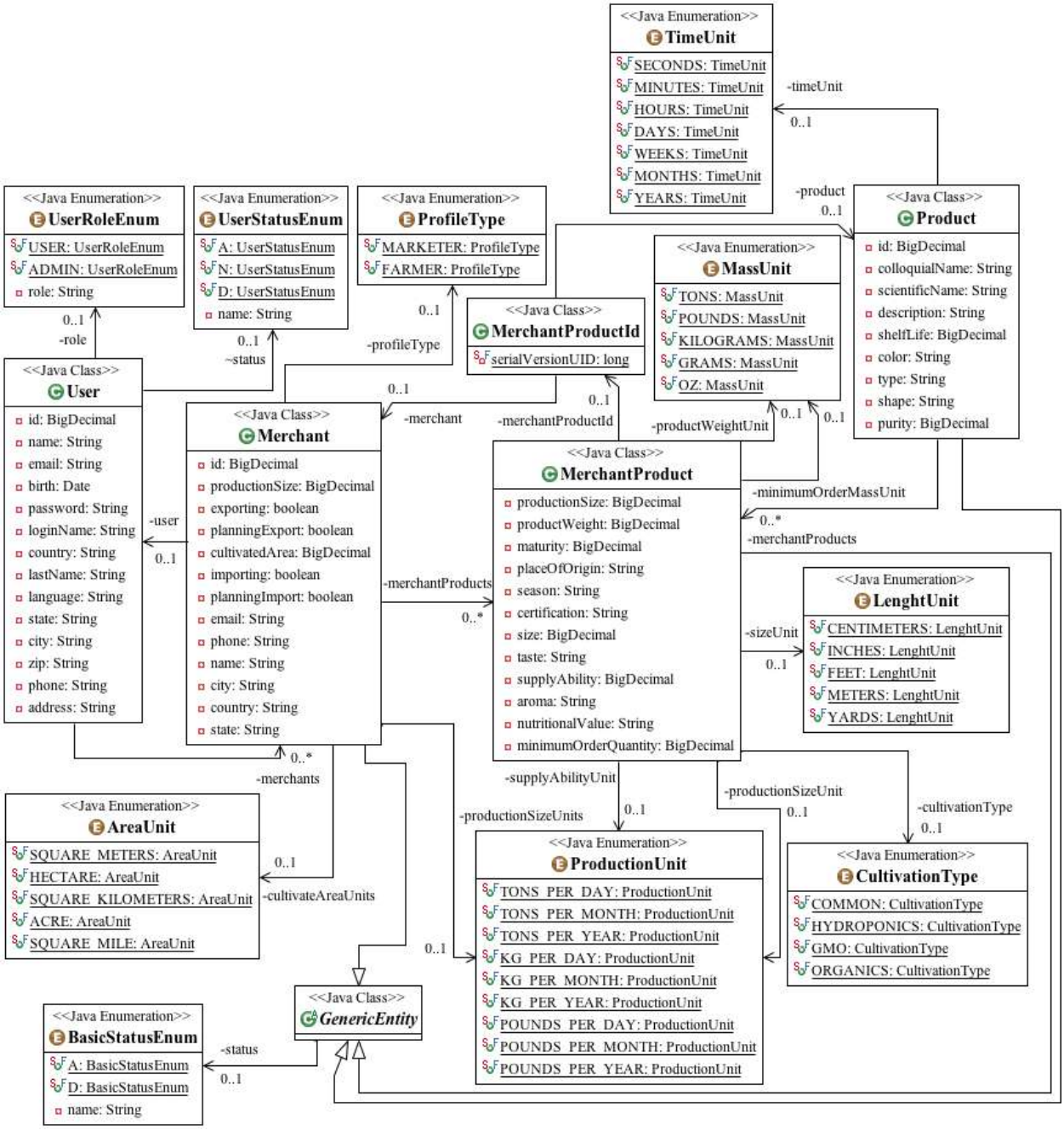


Figura 5.12 Diagrama de clases para el Módulo 1.

La Figura 5.12 muestra las asociaciones y relaciones entre clases que componen al Módulo 1, las cuales se listan a continuación (solo se listan las clases relevantes que definen al módulo, las demás son Enumeraciones que representan listas de constantes de valores usados en la aplicación, como unidades de medida, tipo de cultivo, etc.):

1. **User**. Representa al usuario de la aplicación, contiene información básica para la creación de un usuario dentro de la aplicación.

2. **Merchant**. Representa un perfil, existen dos tipos de perfil: Productor y Comercializador, así que esta clase modela a ambos perfiles, para distinguir entre comercializador o productor la clase contiene un atributo *profileType* que indica que tipo de perfil es.
3. **Product**. Esta clase modela al producto agrícola.
4. **MerchantProduct**. Esta clase modela las asociaciones entre Producto y el perfil Comercializador y Producto y el perfil Productor. En otras palabras indica que producto corresponde a cual productor y a cual comercializador, además de las características que hacen diferente el producto de un usuario a los demás productos de otros usuarios.
5. **MerchanProductId**. Esta clase es la equivalencia en objeto de la llave primaria compuesta, para la clase MerchantProduct.

Las siguientes son las asociaciones existentes en el Módulo 1:

1. **User-Merchant**. Un usuario puede tener varios perfiles.
2. **Merchant-User**. Un perfil solo puede estar relacionado a un solo usuario.
3. **Merchant-MerchantProduct**. Un perfil puede tener varios productos asociados a él, en virtud de que un usuario haya dado de alta varios productos.
4. **Product-MerchantProduct**. Sucede lo mismo que el punto anterior para el caso de los productos.
5. **MerchantProduct-MerchantProductId**. Un MerchantProduct puede tener solo un id.
6. **MerchantProductId-Merchant**. El id relaciona solamente un perfil a un solo producto.
1. **MerchantProductId-Product**. Lo mismo que el punto anterior.

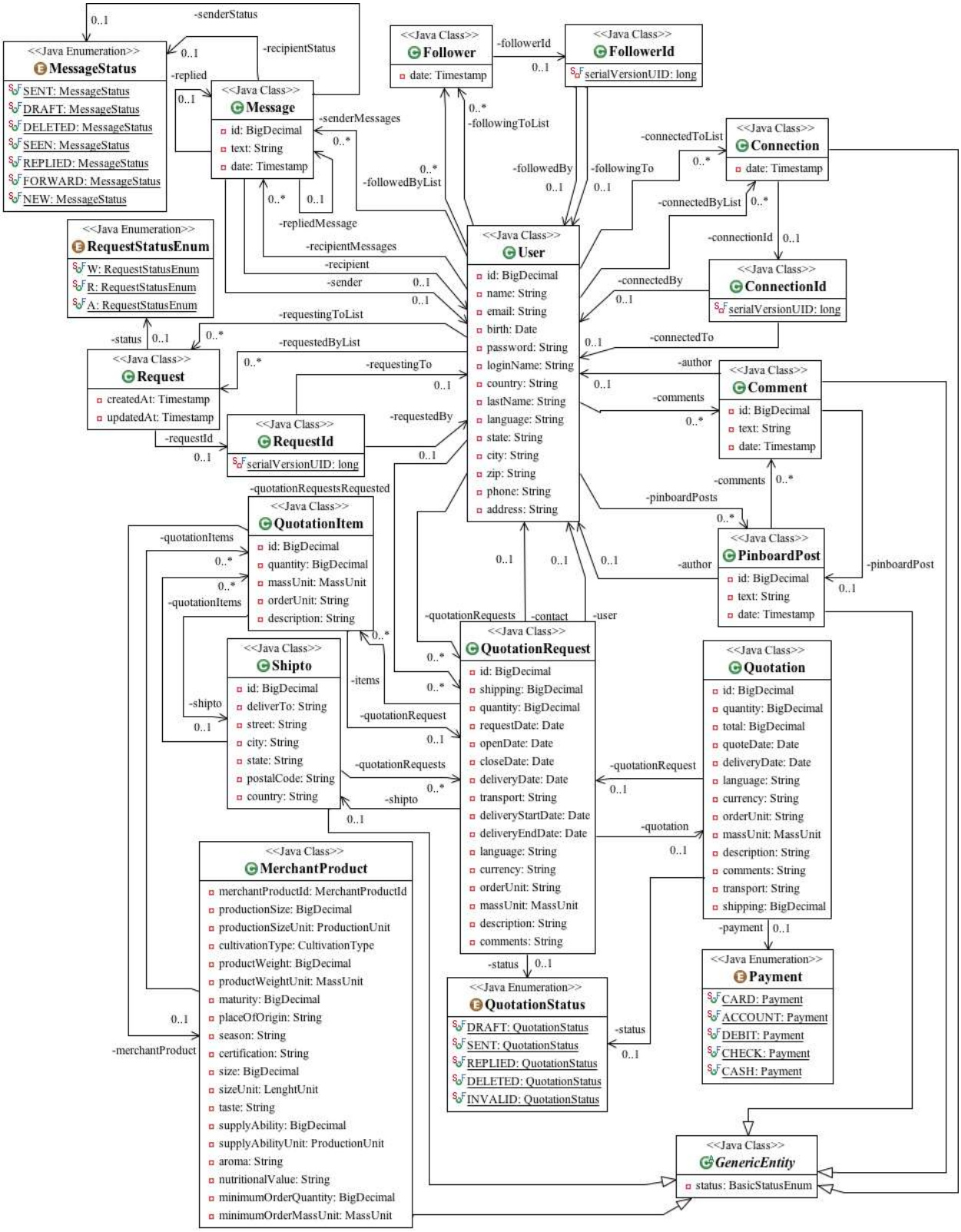


Figura 5.13 Diagrama de clases para el módulo 2.

La Figura 5.13 muestra las asociaciones y relaciones entre clases para el Módulo 2, las cuales se detallan a continuación:

1. **Message**. Modela la estructura de los mensajes que son enviados entre usuarios.
2. **Follower**. Esta clase modela la interacción social de seguimiento entre usuarios.
3. **FollowerId**. Clase que representa la llave primaria compuesta para la clase Follower.
4. **Request**. Es una solicitud de solicitud de conexiones entre usuarios.
5. **RequestId**. Clase que representa la llave primaria compuesta para la clase Request.
6. **Connection**. Modela la interacción social de conexión entre usuarios.
7. **ConnectionId**. Clase que representa la llave primaria compuesta para la clase Connection.
8. **PinboardPost**. Modela las publicaciones del usuario en su Pinboard.
9. **Comment**. Clase que modela los comentarios que los usuarios hacen en los pinboardPosts de otros usuarios.
10. **QuotationRequest**. Modela las solicitudes de cotización entre usuarios.
11. **QuotationItem**. Modela los "items" o productos que un usuario pide cotizar al usuario relacionado con éstos.
12. **Quotation**. Es la respuesta a la solicitud de cotización de un usuario.
13. **Shipto**. Detalla la información de envío.

Las asociaciones entre clases son las siguientes:

1. **User-Message** 1. Un usuario tiene una lista de mensajes que ha recibido.
2. **User-Message** 2. Un usuario tiene una lista de mensajes que ha enviado.
3. **Message-Message** 1. Un Message tiene un Message que representa un mensaje en respuesta a este mensaje.
4. **Message-Message** 2. Un Message tiene un Message que representa un mensaje al que este mensaje responde.
5. **Message-User** 1. Un mensaje tiene un usuario que representa a quien envió el mensaje
6. **Message-User** 2. Un mensaje tiene un usuario que representa a quien será enviado el mensaje.
7. **User-Follower** 1. Un User tiene una lista de Followers que representa los usuarios a los que sigue.
8. **User-Follower** 2. Un User tiene una lista de Followers que representa los usuarios por los que es seguido.
9. **Follower-FollowerId**. Un Follower tiene un solo FollowerId.
10. **FollowerId-User** 1. El FollowerId tiene un usuario que sigue a otro User.
11. **FollowerId-User** 2. El FollowerId tiene un usuario que es seguidor por otro User.
12. **User-PinboardPost**. User tiene una lista de PinboardPosts que son las publicaciones hechas por el usuario.

13. **PinboardPost-User**. PinboardPost tiene un User que representa al autor.
14. **PinboardPost-Comment**. PinboardPost tiene una lista de Comments que son los mensajes que otros usuarios publican en la publicación.
15. **Comment-PinboardPost**. Un comment tiene un solo PinboardPost, un comentario para una sola publicación.
16. **User-Comment**. Un User tiene una lista de Comments, son todos los comentarios hechos por el usuario.
17. **Comment-User**. Un Comment tiene un User que representa al autor del comentario.
18. **User-Request** 1. Un User tiene una lista de Requests que son las solicitudes de conexión que el usuario hace a otros usuarios.
19. **User-Request** 2. Un User tiene una lista de Requests que son las solicitudes de conexión que otros usuarios le envían.
20. **Request-RequestId**. Un Request tiene solo un requestId.
21. **RequestId-User** 1. Un RequestId tiene un User que representa al usuario que solicita conexión.
22. **RequestId-User** 2. Un RequestId tiene un User que representa al usuario al que se le solicita la conexión.
23. **User-Connection** 1. Un User tiene una lista de Connections que son los usuarios a los que está conectado.
24. **User-Connection** 2. Un User tiene una lista de Connections que son los usuarios que están conectados con él.
25. **Connection-ConnectionId**. Un Connection tiene un solo ConnectionId.
26. **ConnectionId-User** 1. Un ConnectionId tiene un solo User que es el usuario que está conectado a otro usuario.
27. **ConnectionId-User** 2. Un ConnectionId tiene un solo User que es el usuario al que otro usuario está conectado.
28. **User-QuotationRequest** 1. Un User tiene una lista de QuotationRequests que son las solicitudes de cotizaciones enviadas al usuario por otros usuarios.
29. **User-QuotationRequest** 2. Un User tiene una lista de QuotationRequests que son las solicitudes de cotizaciones que el usuario envía a otros usuarios.
30. **QuotationRequest-User** 1. Un QuotationRequest tiene un User que representa al usuario que hizo la solicitud de cotización.
31. **QuotationRequest-User** 2. Un QuotationRequest tiene un User que representa al usuario al que se le solicita una cotización.
32. **QuotationRequest-QuotationItem**. Un QuotationRequest tiene una lista de QuotationItems que representa los productos que se piden cotizar.
33. **QuotationItem-QuotationRequest**. Un QuotationItem tiene un solo QuotationRequest un item está relacionado con solo una solicitud de cotización.
34. **QuotationItem-MerchantProduct**. Un QuotationItem tiene un MerchantProduct un item está relacionado a una sola asociación entre el perfil y el producto.

35. **MerchantProduct-QuotationItem**. Un MerchantProduct tiene una lista de QuotationItems, en virtud de que la asociación entre el perfil y el producto pueda ser contenida en muchas solicitudes de cotización.
36. **QuotationItem-Shipto**. Un QuotationItem tiene un solo Shipto.
37. **Shipto-QuotationItem**. Un Shipto tiene una lista de QuotationItems.
38. **QuotationRequest-Shipto**. Un QuotationRequest tiene un Shipto.
39. **Shipto-QuotationRequest**. Un Shipto tiene una lista de QuotationRequests.
40. **QuotationRequest-Quotation**. Un QuotationRequest tiene un solo Quotation.
41. **Quotation-QuotationRequest**. Un Quotation tiene un solo QuotationRequest.

El modelo de clases descrito antes es mapeado al modelo relacional que Hibernate crea en base a la jerarquía de clases, en la Figura 5.14 se puede apreciar el esquema de Base de Datos generado.

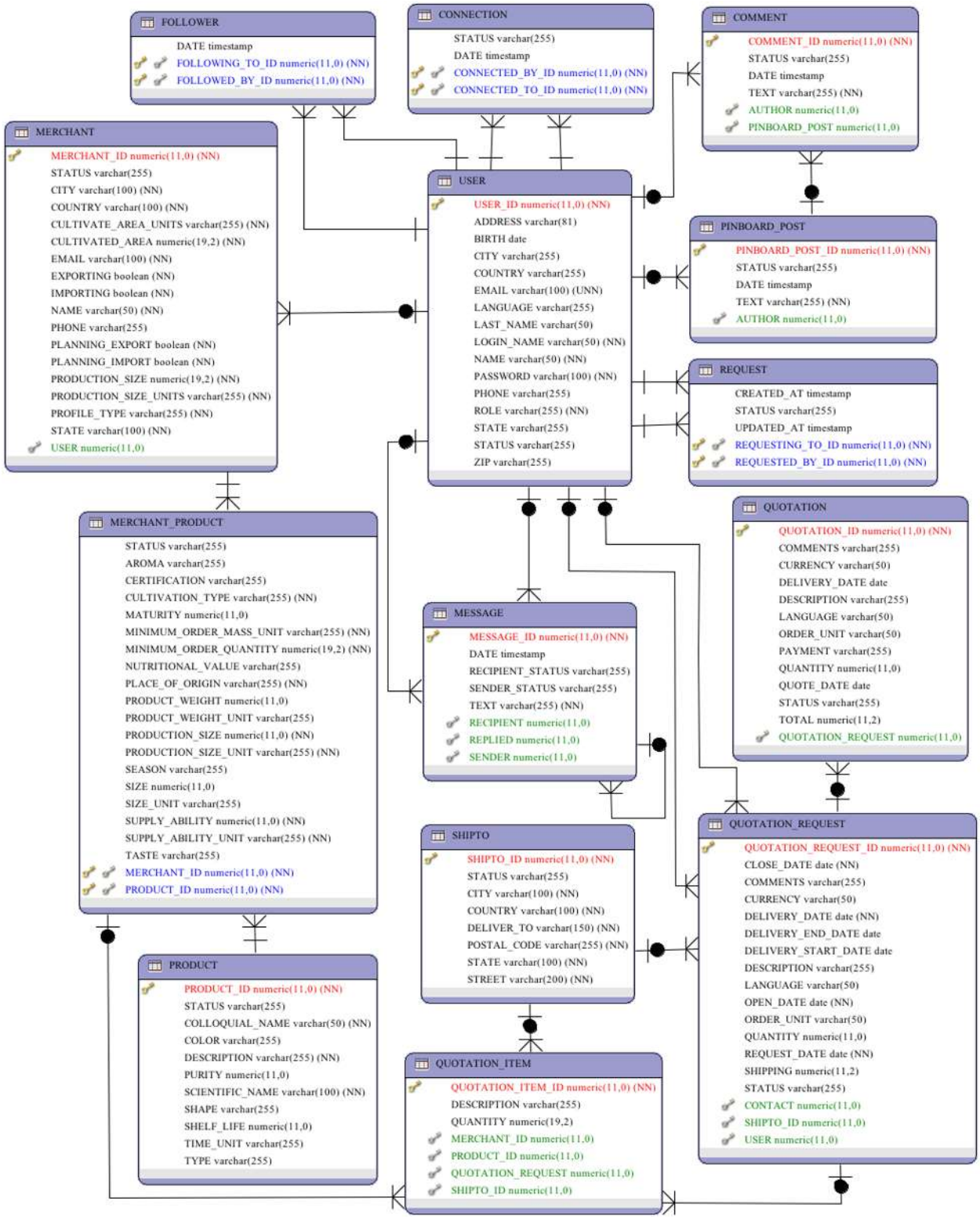


Figura 5.14 Diagrama E-R por notación IE de la base de datos generada por Hibernate.

Las tablas que produce Hibernate al mapear de un modelo a otro y su equivalencia en clases son

las mostradas en la Tabla 5.4.

Tabla 5.4 Equivalencia de tablas y clases.

	Tabla modelo E-R	Equivalencia en clase POO
1	FOLLOWER	Follower
2	CONNECTION	Connection
3	COMMENT	Comment
4	MERCHANT	Merchant
5	USER	User
6	PINBOARD_POST	PinboardPost
7	REQUEST	Request
8	MERCHANT_PRODUCT	MerchantProduct
9	MESSAGE	Message
10	QUOTATION	Quotation
11	PRODUCT	Product
12	SHIPTO	Shipto
13	QUOTATION_REQUEST	QuotationRequest
14	QUOTATION_ITEM	QuotationItem

En este capítulo se mostró todo lo concerniente al diseño de la aplicación, los patrones de diseño empleados para su desarrollo, los módulos que conforman a la aplicación y qué acciones desempeña cada módulo así como su propósito, se mostró también el flujo que sigue la aplicación y sus componentes, además del modelo de datos y la estructura de errores manejada en la aplicación. Algo importante que debe ser dicho es que el modelo de datos mostrado en este capítulo en la Figura 5.12, Figura 5.13 y Figura 5.14 es solo un modelo de datos inicial que después al ser utilizado probablemente tenga que cambiar. El desarrollo de software es un proceso de mejora continua, las necesidades de los usuarios cambian y por ello el software debe evolucionar para adaptarse y cumplir esas necesidades.

En el siguiente capítulo se muestran los resultados obtenidos en el desarrollo de la aplicación.

Capítulo 6 Resultados

En este capítulo se muestran el total de servicios web implementados y los resultados obtenidos en los “test cases” realizados en dichos servicios haciendo uso de herramientas para cobertura de código (code coverage).

La cobertura de código es un proceso por medio del cual se encuentran áreas de un programa que no están siendo probadas por los test cases, de esa manera se deben de crear tests adicionales para incrementar la cobertura de código. El hecho de que partes del código no estén siendo cubiertas por ningún tipo de test puede llevar a defectos potenciales en la etapa de producción. La cobertura de código es una medida cuantitativa de que tan bueno es un test case. Además el análisis de cobertura de código identifica test cases redundantes que no incrementan la cobertura e indica que el código está siendo ejecutado una vez que los test cases corren. La cobertura de código se usa para asegurar la calidad de los tests cases.

6.1 Servicios implementados

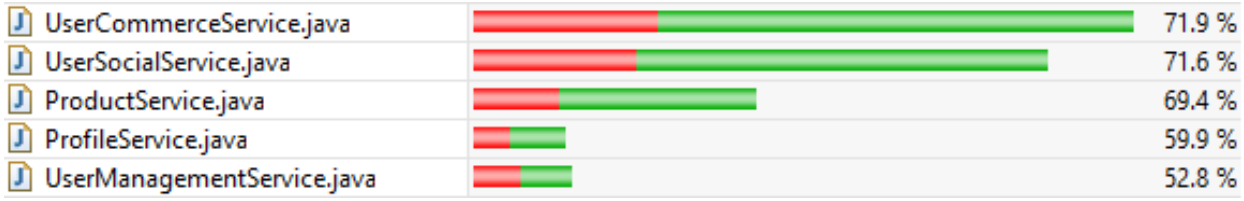
Para los Módulos 1 y 2 se desarrollaron un total de 153 “test cases”, muchos de esos “test cases” prueban servicios de mapeo de DTOs a entidades y viceversa, dichos servicios no se mostrarán como parte de los servicios implementados debido a que no tienen un endpoint, por lo tanto, no es posible accesarlos desde fuera de la aplicación, son solo de uso interno.

El total de servicios web desarrollados para los Módulos 1 y 2 se encuentran listados en los Apéndices: Apéndice B Servicios REST implementados para las entidades Product y MerchantProduct, Apéndice C Servicios REST implementados para la entidad Merchant, Apéndice D Servicios REST implementados para la entidad User y entidades relacionadas (Post, Follower, Connection, Comment, Message, QuotationRequest, Quotation, Shipto). Algo que debe remarcarse es que un endpoint de un servicio web puede corresponder no solo a un solo servicio sino a un conjunto de ellos, dependiendo de los parámetros que sean enviados al servicio web se decide qué servicio debe ser llamado. En el caso del Módulo 3 los servicios web desarrollados se encuentran en el Apéndice E Servicios implementados para el módulo de repositorio de documentos.

6.2 Resultados de la cobertura de código

Los servicios web listados en los apéndices correspondientes a los Módulos 1 y 2 están contenidos en los archivos mostrados en la Figura 6.1. El análisis de cobertura de código se realizó por medio de la herramienta EclEmma en el IDE Eclipse.

Figura 6.1 Cobertura de código de los archivos que contienen los servicios desarrollados para los Módulos 1 y 2.



Se muestra el nombre del archivo, una gráfica que indica la proporción de tamaño entre los archivos analizados y una comparación entre código con cobertura (verde) y sin (rojo), por último, se muestra el porcentaje de código cubierto por los tests.

La lógica de negocio de los servicios web en el Apéndice B Servicios REST implementados para las entidades Product y MerchantProduct está contenida en el archivo ProductService.java, mientras que, en el caso del Apéndice C Servicios REST implementados para la entidad Merchant se encuentra en el archivo ProfileService.java, por último, en el caso del Apéndice D Servicios REST implementados para la entidad User y entidades relacionadas (Post, Follower, Connection, Comment, Message, QuotationRequest, Quotation, Shipto) se encuentra en los archivos UserManagementService.java, UserSocialService.java y UserCommerceService.java. Un punto importante que es necesario establecer es que para el caso de los Módulos 1 y 2 se probó el flujo normal de la aplicación, es decir, sin tomar en cuenta los casos en los que se dan excepciones o errores, es por ello que los porcentajes de cobertura de código son en algunos casos debajo del 70%.

Para el caso de la cobertura de código de los servicios que contienen la lógica de negocio de los servicios web implementados para el Módulo 3 mostrados en el Apéndice E Servicios implementados para el módulo de repositorio de documentos se obtuvo el resultado que se muestra en la Figura 6.2.

Figura 6.2 Cobertura de código del archivo que contiene los servicios desarrollados para el módulo 3.

Se muestra el nombre del archivo, la gráfica que indica el porcentaje de código cubierto en color verde y el porcentaje de código no cubierto en color rojo, así también, se muestra el porcentaje de código cubierto por los tests.

El casi 20% de código no cubierto mostrado en la Figura 6.2 son casos en los cuales los test cases no cubren los bloques try-catch que involucran errores de acceso a la base de datos.

Eventualmente estas partes de código no cubiertas serán cubiertas en los test cases con el fin de lograr el 100% de cobertura.

Capítulo 7 Conclusiones y trabajo futuro

Esta Tesis, como una tesis de ingeniería, busca resolver un problema actual usando tecnología actual con el fin de mejorar las condiciones que se dan hoy en día en el negocio agrícola, teniendo como objetivo el innovar los medios de negociación actuales que se desarrollan entre los diferentes participantes en el negocio agrícola, para ello se desarrolló un REST API para una plataforma de red social enfocada en el negocio agrícola.

Este REST API está compuesto de 3 módulos: Módulo de Usuarios, Módulo de Acciones Sociales y E-commerce y el Módulo de Repositorio de Documentos. Se explicó su funcionamiento y qué problemas resuelve cada uno estos módulos, además de, cómo fueron modelados y las tecnologías utilizadas para el desarrollo de los mismos.

Si bien los módulos desarrollados para esta Tesis brindan las funcionalidades básicas que una plataforma de red social de este tipo requiere, aún existen muchas más funcionalidades y mejoras que pueden ser desarrolladas como trabajo futuro para este proyecto y que se espera se desarrollen como parte de un proyecto más grande y ambicioso.

1. **Módulo de Interacción con Redes Sociales**. Agregar interacción con las redes sociales más populares como lo son Facebook y Twitter, además de habilitar la funcionalidad para envío de correos electrónicos a los usuarios sobre notificaciones, mensajes, etcétera, de forma que, los usuarios estén al tanto de lo que ocurre con sus contactos o seguidores no solo dentro de la aplicación sino también fuera de ella.
2. **Módulo de Publicidad**. Este módulo servirá para proporcionar servicios publicitarios a empresas interesadas en hacer promocionar sus productos a los usuarios de la red social. La publicidad puede ser una manera por medio de la cual se puedan obtener ingresos monetarios.
3. **Cliente web**. Desarrollar un cliente web con interfaz gráfica de usuario UI que consuma el REST API desarrollado. El desarrollo de un cliente web que haga uso de los servicios web proporcionados por la aplicación es de vital importancia puesto que abriría el acceso a la aplicación a una importante cantidad de personas del sector agrícola que es justamente lo que se busca.
4. **Apps para dispositivos móviles**. Desarrollar aplicaciones móviles para Android, iOS y demás sistemas operativos en auge para dispositivos móviles. Hoy en día la presencia de las redes sociales es total ya no hay restricción únicamente a clientes web, ahora también existen aplicaciones para dispositivos móviles e incluso aplicaciones de escritorio que nos mantienen en contacto con los demás usuarios de las plataformas de redes sociales.
5. **Más funcionalidades E-Commerce**. Agregar mayor número de funcionalidades e-commerce como transacciones comerciales en línea. De manera que se integren cada vez más funcionalidades a la plataforma hasta alcanzar un estado en que el usuario pueda

realizar el proceso completo de compra y venta de productos agrícolas dentro de la misma plataforma.

Por supuesto que existen muchas más características que pueden ser agregadas como parte de la aplicación, pero, al menos para la continuación y crecimiento de un proyecto como éste lo anterior es suficiente para alcanzar una plataforma funcional que pueda ser utilizada a lo largo de todo el mundo.

Este trabajo ha servido al autor para aplicar los conocimientos adquiridos como Ingeniero en Computación y además aprender sobre un campo de aplicación, en este caso el negocio agrícola y alimentario. Esta aplicación servirá para motivar el desarrollo económico y tecnológico de los productores agrícolas y tiene un gran potencial para crecer en otros sentidos como servicios de publicidad y apoyo a la logística.

Al momento de término de esta Tesis el REST API desarrollado no ha sido liberado al público y no se cuenta con usuarios. Lo que más desea el autor es que este proyecto vea la luz, crezca y prospere, que cumpla con su objetivo de facilitar los procesos en el negocio agrícola por medio de la retroalimentación de los usuarios, una vez que sea liberado, para de esa forma poder mejorar los aspectos que así lo requieran.

Referencias

(s.f.). Obtenido de Introducing JSON: www.json.org

(2011). Obtenido de openTRANS: http://www.opentrans.de/en

Alex, B., & Taylor, L. (s.f.). Obtenido de Spring Security Reference Documentation: http://docs.spring.io/spring-security/site/docs/3.0.x/reference/springsecurity.html

Alexa Internet, Inc. (2014). *Top Sites*. Obtenido de Alexa: http://www.alexa.com/topsites

Ariba, Inc. (2014). Obtenido de Ariba: http://www.ariba.com

Backstrom, L., Boldi, P., Rosa, M., Ugander, J., & Vigna, S. (2012). Four Degrees of Separation.

Barabási, A.-L., & Bonabeau, E. (2003). Scale-Free Networks. *Scientifica American vol. 288, no. 5*, 50-59.

Barabási, A.-L., & Réka, A. (1999). Emergence of Scaling in Random Networks. *Science vol. 286*, 509-512.

Boyd, D. M., & Ellison, N. B. (2008). Social Network Sites: Definition, History, and Scholarship. *Journal of Computer-Mediated*.

cXML.org. (s.f.). Obtenido de cXML: http://cxml.org/

Faber, S. (2012). Obtenido de Mockito: https://code.google.com/p/mockito/

Facebook. (2014). Obtenido de Facebook Developers: https://developers.facebook.com/docs/graph-api/reference/v2.0/user

Fielding, R. T. (2000). Obtenido de Architectural Styles and the Design of Network-based Software Architectures: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Food and Agriculture Organization of the United Nations. (2013). *FAO Statistical Year Book 2013 World Food and Agriculture.*

Freeman, E., Freeman, E., Sierra, K., & Bates, B. (2004). *Head First Design Patterns.*

Gierke, O., Darimont , T., & Strobl, C. (2014). Obtenido de Spring Data JPA - Reference Documentation: Véase http://docs.spring.io/spring-data/jpa/docs/1.6.0.RELEASE/reference/html/jpa.repositories.html

GoPivotal. (2014). *Spring Data*. Obtenido de Spring: http://projects.spring.io/spring-data

GoPivotal, Inc. (2014). Obtenido de Spring Framework: http://projects.spring.io/spring-framework/

Interagency Agricultural Projections Committee. (2014). *USDA Agricultural Projections to 2023.* USDA Agricultural Projections.

Johnson, R., Hoeller, J., Donald, K., & Sampaleanu, C. (2014). Obtenido de Spring Framework Reference Documentation: http://docs.spring.io/spring/docs/4.0.5.RELEASE/spring-framework-reference/html/beans.html

Keith, M., & Schincariol, M. (2009). *Pro JPA 2.* Apress.

Kleinberg, J. M. (2000). Navigation in a small world. *MacMillan Magazines*, 845.

Konda, M. (s.f.). *Just Spring.* O'Reilly.

Kuchana, P. (2004). *Software Architecture Design Patterns in Java.*

Milgram, S. (1967). The small world problem. *Psychology*, 60-67.

Milgram, S., & Travers, J. (1969). An experimental study of the small world problem. *Sociometry, vol 32, no. 4*, 425-443.

MongoDB, Inc. (2013). Obtenido de mongoDB: http://www.mongodb.org/

Newman, M. E. (2003). Thes structure and function of complex networks. *SIAM Review 45*, 167-256.

OECD-FAO. (2014). *Agricultural Outlook 2013-2022*. Obtenido de OECD-FAO Agricultural Outlook: http://www.oecd.org/site/oecd-faoagriculturaloutlook/

Organización de las Naciones Unidas para la Agricultura y la Alimentación. (2000). *El estado Mundial de la Agricultura y la Alimentación 2000.*

Pedrero, M. (2001). *Censos Agropecuarios y Género - Conceptos y Metodología.* Organización de las Naciones Unidas para la agricultura y la alimentación.

People Soft. (2006). Obtenido de Associate Connection: https://associateconnection.staples.com

Pivotal. (2014). *Spring MVC framework*. Obtenido de Spring Framework Reference: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html

ProMéxico. (Marzo de 2010). *Cómo Participar con éxito en Ferias y Exposiciones Internacionales.*

Red Hat. (2014). *Hibernate ORM*. Obtenido de Hibernate: http://hibernate.org/orm/

Red Hat. (2014). *What is Object/Relational Mapping?* Obtenido de Hibernate: http://hibernate.org/orm/what-is-an-orm/#the-object-relational-impedance-mismatch

Réka, A., & Barabási, A.-L. (2002). Statistical Mechanics of Complex Networks. *Reviews of Modern Physics*.

Schmitz, V., Kelkar, O., Otto, B., & Weiner , N. (2009). Obtenido de openTRANS: http://www.opentrans.de/en

Stone, G. (2005). Agribusiness Role in Extension, Education and Training: A case study. *RIRDC Publication no. 05/086*.

Strogatz, S. H., & Watts, D. J. (1998). Collective Dynamics of 'small-world' networks. *Nature vol. 393*, 440-442.

Tahchiev, P., Leme, F., Massol, V., & Gregory, G. (s.f.). *JUnit in Action.* Manning.

Tata Consultancy Services. (2014). *Ultimatix*. Obtenido de Ultimatix: https://www.ultimatix.net/

The hsql Development Group. (2014). Obtenido de HyperSQL: http://hsqldb.org/

Toriumi, F., Okada, I., Yamamoto, H., Suwa, H., Izumi, K., & Hashimoto, Y. (2011). Clasification of Social Network Sites based on Network Indexes and Communication Patterns.

Twitter. (2013). Obtenido de Twitter Developers: https://dev.twitter.com/docs/api/1.1/get/statuses/user_timeline

U.S. Department of Commerce, U.S. Census Bureau. (2013). *International Data Base*. Obtenido de United States Census Bureau: http://www.census.gov/population/international/data/idb/informationGateway.php

Apéndice A Errores manejados por la aplicación

Error	HTTP	Interno	Mensaje
REST_API_DOC		00000	
RESOURCE_NOT_FOUND	404	00001	The requested resource was not found
UNKNOWN_ERROR	500	00002	Unknown error found
ERROR_IN_APPLICATION	500	00003	Error in the application
ERROR_IN_SERVER	500	00004	Error in business service
EMPTY_RESULT	404	00005	No result found with that criteria
NULL_INPUT	400	00006	Null input received, expecting: {0}
UNAUTHORIZED	401	00007	Unauthorized resource
NOT_VALID_RFQ	400	10001	Error while trying to validate request for quotation
NOT_VALID_DATE_RANGE	400	10002	Error while trying to validate date range
NOT_VALID_TIMESTAMP_RANGE	400	10003	Error while trying to validate timestamp range
INVALID_QUERY_PARAM_SET	400	20001	Invalid set of query parameters
DATA_ACCESS_ERROR	500	30001	There was an error while accessing database
USER_NOT_FOUND	500	30002	The user was not found
PROFILE_NOT_FOUND	500	30003	The profile was not found
PRODUCT_NOT_FOUND	500	30004	The product was not found
MARKETER_PRODUCT_NOT_FOUND	500	30005	The marketerProduct was not found
FOLLOWER_NOT_FOUND	500	30006	The follower was not found

PINBOARD_POST_NOT_FOUND	500	30007	The pinboard post was not found
CONTACT_NOT_FOUND	500	30008	The contact was not found
COMMENT_NOT_FOUND	500	30009	The comment was not found
REQUEST_NOT_FOUND	500	30010	The request was not found
CONNECTION_NOT_FOUND	500	30011	The connection was not found
REQUEST_ACCEPTED	400	30012	The request has been accepted
REQUEST_WAITING	400	30013	The request has been sent, but it has not been accepted yet
REQUEST_NOT_CORRECT	500	30014	The request was not correct
REQUEST_REJECTED	400	30015	The request has already been rejected
MESSAGE_NOT_FOUND	500	30016	The message was not found
QUOTATION_REQUEST_RESPONDED	400	30017	The request for quotation has been responded
QUOTATION_REQUEST_NOT_FOUND	500	30018	The request for quotation has not been found
SHIPTO_NOT_FOUND	500	30019	The ship to information was not found
QUOTATION_ITEM_NOT_FOUND	500	30020	The item was not found
QUOTATION_REQUEST_EXPIRED	400	30021	The request for quotation has expired
QUOTATION_NOT_FOUND	500	30022	The quotation has not been found
MERCHANT_NOT_FOUND	500	30023	The merchant has not been found
MERCHANT_PRODUCT_NOT_FOUND	500	30024	The merchant product has not been found
QUOTATION_REQUEST_NOT_VALID	400	30025	The request for quotation is

			not valid
QUOTATION_NOT_VALID	400	20036	The quotation is not valid
QUOTATION_ITEM_NOT_VALID	400	30027	The quotation item is not valid
PRODUCT_NOT_DELETED	400	30028	The product could not be deleted because there are objects depending on it
SHIPTO_NOT_UPDATED	400	30029	The shipto could not be updated because there are objects depending on it
SHIPTO_NOT_DELETED	400	30030	The shipto could not be deleted because there are objects depending on it
FILE_NOT_FOUND	500	30031	The file was not found
FILE_ALREADY_EXISTS	400	30032	The file already exists
MAPPING_ERROR	500	30033	There was a mapping error {0} to {1}
FATAL_ERROR	500	99999	Fatal error

Apéndice B Servicios REST implementados para las entidades Product y MerchantProduct

HTTP	URI	Descripción
GET	/product/all	Lista todos los productos por estatus. Requiere el parámetro *status*.
GET	/product/{id}	Trae un producto por su id.
POST	/product/new	Crea un producto dentro de la base de datos con la información pasada como parámetro (productPartialDto) al servicio en un objeto JSON.
POST	/product	Actualiza la información de un Product en la base de datos. Requiere de un objeto JSON *productPartialDto* en el cuerpo de la petición HTTP.
DELETE	/product/{id}	Da de baja un producto por medio del cambio de su status de activado a desactivado.
GET	/product/active/{id}	Activa un producto dado de baja por su id.
POST	/product/merchant	Crea un MerchantProduct dentro de la base de datos. Requiere un objeto JSON como parámetro (*merchantProductPartialDto*)
DELETE	/product/merchant/{merchantId}, {productId}	Da de baja un MerchantProduct.
PUT	/product/merchant	Actualiza la información de un MerchantProduct. Requiere un objeto JSON como parámetro (*merchantProductPartialDto*).
PUT	/product/merchant/active	Activa un MerchantProduct que ha sido dado de baja. Requiere un objeto JSON como parámetro (*MerchantProductPartialDto*).
GET	/product/merchant	Trae de la base de datos un MerchantProduct, para ello se deben especificar dos parámetros: *merchantId* y *productId*
GET	/product/merchants	Trae una lista de MerchantProducts, se

		pueden especificar 3 parámetros como opción *merchantId*, *productId* o *status.*

Apéndice C Servicios REST implementados para la entidad Merchant

HTTP	URI	Descripción
GET	/merchant	Trae un Merchant de la base de datos por el id especificado (*merchantId*)
GET	/merchants/all	Trae una lista de Merchants. Es posible especificar varios parámetros opcionales: *profileStatus* y *profileType.*
POST	/merchant/new	Crea un Merchant dentro de la base de datos, requiere de un objeto JSON como parámetro (*MerchantPartialDto*).
PUT	/merchant	Actualiza la información de un Merchant, requiere de un objeto JSON como parámetro (*MerchantPartialDto*)
DELETE	/merchant/{merchantId}	Elimina un Merchant.
GET	/merchant/active/{merchantId}	Activa un Merchant que ha sido eliminado.

Apéndice D Servicios REST implementados para la entidad User y entidades relacionadas (Post, Follower, Connection, Comment, Message, QuotationRequest, Quotation, Shipto)

HTTP	URI	Descripción
GET	/user	Trae un usuario de la base de datos por id especificando el parámetro *id*.
GET	/user/all	Trae una lista de todos los usuarios, se puede especificar el parámetro *status*.
POST	/user	Actualiza la información de un usuario en la base de datos, requiere de un objeto JSON *user*.
POST	/user/new	Crea un nuevo usuario dentro de la base de datos, requiere de un objeto JSON *user*.
GET	/user/active/{id}	Activa un usuario que ha sido recién creado o que había sido desactivado.
DELETE	/{id}	Cambia el estatus del usuario a eliminar.
POST	/user/password/update	Actualiza el password del usuario, requiere de un objeto JSON *userPassword*.
GET	/user/password/reset/{id}	Resetea el password del usuario. Se reemplaza por un password generado automáticamente.
POST	/user/post	Crea un PinboardPost. Requiere de un objeto JSON *pinboardPostPartialDto*.
GET	/user/post/{id}	Trae una lista de PinboardPosts. Se pueden especificar varios parámetros: *id* que es el id del autor, *find.mix* opcional para mezclar los posts de los usuarios a los que sigue y los suyos y *find.others* para mostrar únicamente posts de usuarios a los que sigue y *status* de los posts. Por default trae solo posts del usuario.
GET	/user/post/{id}	Trae un pinboardPost por su id.
DELETE	/user/post/{postId}	Da de baja un pinboardPost.

PUT	/user/follow	Seguir a un usuario, requiere de un objeto JSON *followerDto*.
GET	/user/followers	Trae todos los seguidores de un usuario por su id, requiere el parámetro *id*.
GET	/user/followed	Trae todos los usuarios seguidos por el usuario. Requiere el id del usuario como parámetro *id*.
DELETE	/user/follow/{followedById},{followingToId}	Deja de seguir a un usuario.
POST	/user/connection/request	Mandar solicitud de conexión con otro usuario, requiere de un objeto JSON *requestDto*.
GET	/user/connection/requesting/{id}	Trae la lista de usuarios a los que el usuario especificado por el id está solicitando por conexión, se puede especificar el parámetro *status* que se refiere al estatus de la solicitud.
GET	/user/connection/requested/{id}	Trae la lista de usuarios que solicitan la conexión al usuario especificado por el id. Se puede especificar el parámetro *status* de las solicitudes.
POST	/user/comment	Crea un comentario en un post. Requiere un objeto JSON *commentDTO*.
GET	/user/comments/{postId}	Trae todos los comentarios de un post especificado por su id. Se puede especificar el parámetro *status* para el estatus de los comentarios.
GET	/user/comment/{commentId}	Trae comentario por su id.
DELETE	/user/comment/{commentId}	Da de baja comentario.
POST	/user/connection	Aceptar solicitud de conexión hecha por usuario. Requiere de un objeto JSON *RequestDto*.
DELETE	/user/connection/request/{requestedById},{requestingToId}	Denegar solicitud de conexión de hecha por otro usuario. Requiere del parámetro *date* que especifica la hora en que se denegó la solicitud.

GET	/user/connections/connectedTo/{id}	Trae la lista de todos los usuarios que aceptaron la solicitud del usuario especificado por su id. Se puede especificar un parámetro *status* que representa el status de los usuarios listados.
GET	/user/connections/connectedBy/{id}	Trae la lista de todos los usuarios que el usuario especificado por el id aceptó. Se puede especificar un parámetro status como en el servicio anterior.
DELETE	/user/connection/{connectedById},{connectedToId}	Cancela la conexión hecha con un usuario. Se debe especificar el parámetro *date* que indica la hora cuando la conexión se dió de baja.
POST	/user/message	Enviar mensaje a un usuario. Requiere un objeto JSON *messagePartialDTO*.
GET	/user/messages	Trae una lista de mensajes. Se pueden especificar una variedad de parámetros: *from* quién mandó el mensaje, *to* a quién se manda el mensaje, *messageId* para indicar la fecha del mensaje a partir de la cual se quieren traer los mensajes, *status* estatus de los mensajes, *sent* para especificar si los mensajes fueron enviados o recibidos.
DELETE	/user/message/{messageId}	Borrar un mensaje por id, requiere el parámetros *userId* que especifica que usuario borra el mensaje si el recipiente o quien lo envía.
GET	/user/message/{messageId}	Cambia el estatus del mensaje a visto.
POST	/user/quotation/request	Manda una solicitud de cotización a un usuario. Requiere de un objeto JSON *quotationRequestDto*.
PUT	/user/quotation/request	Actualiza la información de una solicitud de cotización. Requiere de un objeto JSON *quotationRequestDto*.
DELETE	/user/quotation/request/{quotationRequestId}	Da de baja una solicitud de cotización.
GET	/user/quotation/request/{quotationRequestId}	Trae una solicitud de cotización especificada por su id.

GET	/user/quotation/requests	Trae una lista de solicitudes de cotización. Se puede especificar una variedad de parámetros *userId* que indica el id del usuario que manda la solicitud, *contactId* que indica el usuario al que se manda la solicitud, *status* que indica el estatus de las solicitudes.
POST	/user/quotation	Enviar cotización como respuesta a la solicitud de cotización enviada de forma anterior por otro usuario. Requiere de un objeto JSON *quotationDto*.
GET	/user/quotation/{quotationId}	Trae un quotationRequest.
GET	/user/quotations	Trae una lista de cotizaciones. Se pueden especificar una serie de parámetros: *userId* que indica el usuario que envió las cotizaciones, *contactId* que indica el usuario al que se enviaron las cotizaciones, *status* estatus de las cotizaciones. Por default trae todas las cotizaciones con estatus *SENT*.
DELETE	/user/quotation/{quotationId}	Da de baja una cotización.
POST	/user/shipto	Crea un Shipto. Se debe especificar un objeto JSON *shiptoPartialDto*.
DELETE	/user/shipto/{shiptoId}	Borra un Shipto.
GET	/user/shiptos	Trae una lista de Shiptos por estatus. Se puede especificar el parámetro *status*. Por default trae todos los Shiptos con estatus activo.
GET	/user/shipto/{shiptoId}	Trae un Shipto por su estatus.
PUT	/user/shipto	Actualiza la información de un Shipto en la base de datos. Se debe especificar un objeto JSON shiptoPartialDto.
POST	/user/quotation/request/draft	Guarda el draft de una solicitud de cotización. Ésto es almacenar los datos de la solicitud de cotización en la base de datos y asignarle un estatus *DRAFT*, para ello se requiere un objeto JSON *quotationRequestDto*.

PUT	/user/quotation/request/{quotationRequestId}/clone	Hace un clon de la solicitud de cotización pero no hace copia ni del id de la solicitud ni del estatus según sea éste.
POST	/user/quotation/draft	Guarda el draft de una cotización. Ésto es almacenar los datos de la cotización y asignarle un status *DRAFT*, para ello se requiere un objeto JSON *quotationDTO*.

Apéndice E Servicios implementados para el módulo de repositorio de documentos

HTTP	URL	Descripción
POST	/file/new	Subir archivo, requiere de un archivo JSON que especifica los metadatos para el archivo a subir y también el archivo que será subido.
GET	/file/{id:.*}	Descarga archivo por medio de su id.
GET	/file/metadata/{fileId:.+}	Trae los metadatos del archivo especificado por su id.
GET	/files/metadata/user/{id}	Trae una lista de los metadatos de varios archivos especificados por el id del autor o dueño de los archivos. Se puede especificar el parámetro *status* que representa el estatus de los archivos.
POST	/file	Actualiza archivo. Requiere un archivo JSON con los metadatos actualizados del archivo y el archivo actual.
DELETE	/file/{id:.*}	Borrar archivo por id de archivo.
POST	/profile/user/image/new	Subir imagen de perfil de usuario. Requiere un archivo JSON con metadatos del archivo y la imagen a subir.
GET	/profile/user/{userId}/image	Trae imagen de perfil del usuario.
POST	/profile/user/image	Actualiza imagen de perfil de usuario. Requiere de un archivo JSON que contiene los metadatos del archivo y requiere también la imagen.